SECURE AND RESILIENT
COMPUTING FOR INDUSTRY AND HUMAN DOMAINS

Volume 1
FUNDAMENTALS OF SECURE AND RESILIENT COMPUTING

Edited by Vyacheslav Kharchenko

Concepts, Standards and Methods
for Secure and Resilient Computing

Fundamentals of Formal Methods
for System Security Assessment

Formal and Intellectual Methods
for System Security and Resilience
Assurance

Introduction in Methods
of Post Quantum Computing
Cryptography

Volume 1. FUNDAMENTALS OF SECURE AND RESILIENT COMPUTING

MULTI-
LECTURE
BOOK

SECURE AND RESILIENT COMPUTING FOR INDUSTRY AND HUMAN DOMAINS

VOLUME 1

FUNDAMENTALS
OF SECURE AND RESILIENT COMPUTING

2017

Co-funded by the
Tempus Programme
of the European Union

Co-funded by the
Tempus Programme
of the European Union

Ministry of Education and Science of Ukraine
National Aerospace University n. a. N. E. Zhukovsky
"Kharkiv Aviation Institute"

V. Sklyar, O. Illiashenko, V. Kharchenko, N. Zagorodna, R. Kozak, O. Vambol, S. Lysenko, D. Medzatyi, O. Pomorova

SECURE AND RESILIENT COMPUTING FOR INDUSTRY AND HUMAN DOMAINS.

Fundamentals of security and resilient computing

Multi-book, Volume 1

V. S. Kharchenko eds.

Tempus project
SEREIN 543968-TEMPUS-1-2013-1-EE-TEMPUS-JPCR
Modernization of Postgraduate Studies on Security and Resilience for Human and Industry Related Domains

2017

V. Sklyar, O. Illiashenko, V. Kharchenko, N. Zagorodna, R. Kozak, O. Vambol, S. Lysenko, D. Medzatyi, O. Pomorova. **Secure and resilient computing for industry and human domains. Volume 1. Fundamentals of security and resilient computing** / Edited by Kharchenko V. S. – Department of Education and Science of Ukraine, National Aerospace University named after N. E. Zhukovsky "KhAI", 2017.

Reviewers:

Dr. Peter Popov, Centre for Software Reliability, School of Informatics, City University of London

Prof. Stefano Russo, Consorzio Interuniversitario Nazionale per l'Informatica (Naples, Italy)

Prof. Todor Tagarev, Centre for Security and Defence Management, Institute of Information and Communication Technologies of the Bulgarian Academy of Sciences;

Prof. Jüri Vain, School of Information Technologies, Department of Software Tallinn University of Technology

The first volume of the three volume book called "Secure and resilient computing for industry and human domains" contains materials of the lecture parts of the study modules for MSc and PhD level of education as well as lecture part of in-service training modules developed in the framework of the SEREIN project "Modernization of Postgraduate Studies on Security Resilience for Human and Industry Related Domains"[1] (543968-TEMPUS-1-2013-1-EE-TEMPUS-JPCR) funded under the Tempus programme are given. The book material covers fundamentals issue of secure and resilient computing, in particular, description of related standards, methods of cryptography, software security assurance and post-quantum computing methods review.

The descriptions of trainings, which are intended for studying with technologies and means of assessing security guarantees, are given in accordance with international standards and requirements. Courses syllabuses and description of practicums are placed in the correspondent notes on practicums and in-service training modules.

Designed for engineers who are currently or tend to design, develop and implement information security systems, for verification teams and professionals in the field of quality assessment and assurance of cyber security of IT systems, for masters and PhD students from universities that study in the areas of information security, computer science, computer and software engineering, as well as for lecturers of the corresponding courses.

The materials in the book are given in a form "as is", desktop publishing of this book is available in hard copy only.

[1] *This project has been funded with support from the European Commission. This publication (communication) reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.*

1 STANDARDS FOR SECURITY OF SAFETY CRITICAL SYSTEMS

1.1 Survey of standards in security

At the present security standards are developed by many national and international standardization organizations. The most relevant to safety critical systems are the following security standards sets:

– ISO/IEC 27000 “Information technology – Security techniques – Information security management systems” standards family states requirements to the Information Security Management System (ISMS) independently from type of computer system or organization; this series contains about 40 parts and is an umbrella document for all other documents in security (see Section 1.2);

– ISO/IEC 15408 “Information technology – Security techniques – Evaluation criteria for IT security” establishes Common Criteria to evaluate security functions and assurance techniques for information product (see Section 1.3) [1];

– ISA/IEC 62443 “Security for Industrial Automation and Control Systems” (see Section 1.4);

– The United States National Institute of Standards and Technology (NIST) developed NIST SP 800 series which cover many security issues; formally NIST standards are national but many countries and companies apply it as valuable state-of-the-art requirements; the NIST Cybersecurity Framework (SCF) based on NIST SP 800-53 “Security and Privacy Controls for Federal Information Systems and Organizations” is described in Section 1.5 [2,3];

– Institute of Electrical and Electronics Engineers (IEEE) standards, such as IEEE 1686-2007 “Standard for Substation Intelligent Electronic Devices IED Cybersecurity Capabilities”, IEEE P1711 “Standard for a Cryptographic Protocol for Cybersecurity of Substation Serial Links”, IEEE 1815-2012 “Standard for Electric Power System Communications-Distributed Network Protocol (DNP3)”;

– Standards applicable to specific domains which give details of the above standards requirements; we consider nuclear standard IEC 62645 “Nuclear power plants – Instrumentation and control

systems – Cybersecurity requirements" with associated IEC 62859 "Nuclear power plants – Instrumentation and control systems – Coordination between safety and cybersecurity" and IEC 62988 "Nuclear power plants – Instrumentation and control important to safety – Selection and use of wireless devices".

Also it should be mentioned a lot of activities, performed in different industrial domains by technical and research organizations. The most powerful organizations are working in USA as a part of the continuing effort to provide effective security standards and guidance to federal agencies and their contractors in support of the Federal Information Security Management Act (FISMA). FISMA was signed into law part of the Electronic Government Act of 2002. There are the following organizations, addressing security [4]:

– The USA Department of Energy (DOE) developed the Cybersecurity Capability Maturity Model (C2M2) from the Electricity Subsector Cybersecurity Capability Maturity Model (ES-C2M2) by removing sector specific references and terminology. The ES-C2M2 was developed in support of a White House initiative led by the DOE, in partnership with the Department of Homeland Security (DHS), and in collaboration with private and public sector experts;

– The American Gas Association (AGA), representing energy utility organizations that deliver natural gas customers industries throughout the United States. The AGA 12 series of documents recommends practices designed to protect supervisory control and data acquisition (SCADA) communications against cyber incidents [5];

– The American Petroleum Institute represents members involved in all aspects of the oil and natural gas industry. API 1164 provides guidance to the operators of oil and natural gas pipeline systems for managing SCADA system integrity and security;

– The Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) operates within the National Cybersecurity and Integration Center (NCCIC), a division of the Department of Homeland Security's Office of Cybersecurity and Communications (DHS CS&C). NCCIC/ICS-CERT is a key component of the DHS Strategy for Securing Control Systems. ICS-CERT works with the control systems community to ensure that recommended practices, which are made

available, have been vetted by subject-matter experts in industry before being made publicly available in support of this program [6];

– The North American Electric Reliability Corporation (NERC) mission is to improve the reliability and security of the bulk power system in North America. NERC has issued a set of security standards, named as Critical Infrastructure Protection (SIP), to reduce the risk of compromise to electrical generation resources and high-voltage transmission systems above 100 kV, also referred to as bulk electric systems.

Also there are a lot of non-profit organizations which develop free guidelines and best practices on security issues including the following:

– The Open Web Application Security Project (OWASP) Foundation supports the following projects: OWASP Software Assurance Maturity Model, OWASP Development Guide, OWASP Testing Guide, OWASP Code Review Guide etc.;

– The Institute for Information Infrastructure Protection (I3P) is a consortium of leading national cybersecurity institutions, including academic research centers, government laboratories, and non-profit organizations. It was founded in September 2001 to help meet a well-documented need for improved research and development (R&D) to protect the nation's information infrastructure against catastrophic failures. The institute's main role is to coordinate a national cybersecurity R&D program and help build bridges between academia, industry, and government;

– International Professional Association ISACA (the former Information Systems Audit and Control Association) developed the good-practice framework Control Objectives for Information and Related Technologies (COBIT) which is created for information technology management IT governance. COBIT provides an implementable set of controls over information technology and organizes them around a logical framework of IT-related processes and enablers. COBIT components include process descriptions, control objectives, management guidelines, and maturity models;

– Center for Internet Security (CIS) released Critical Security Controls for Effective Cyber Defense (CSC) framework, which is also known as CIS CSC or CCS CSC. CCS CSC includes he guidelines consist of 20 key actions, called CSC, that organizations should take to block or mitigate known attacks. The controls are designed so that

primarily automated means can be used to implement, enforce and monitor them.

Taking into account variety of security standards, it should be noted they focus on some common issues. These issues include the following:

– Risk Management and Assessment [7];
– Information Security Management System [2,3];
– Security Life Cycle [8];
– Security Levels [4];
– Failures and attack avoidance [9,10];
– Security and safety relation for critical systems [4].

General security concept, directed to comprehensive security assurance, is described in Part 4 of this multi-book.

Below in this section a survey is done for the main security standards, such as ISO/IEC 27000**,** ISA/IEC 62443, and NIST SP 800.

1.2 Standards family ISO/IEC 27000

ISO/IEC 27000 “Information technology – Security techniques– Information security management systems” standards family contains about 40 parts and is an umbrella document for all other documents in security. Now many parts of ISO/IEC 27000 are booming, so many new parts are appearing and some existing parts are reworking once per 3-5 years.

The title standard in the family is ISO/IEC 27000:2016 “Information security management systems – Overview and vocabulary”.

All ISO/IEC 27000 standards family can be divided in the three following sets:

– Standards specifying requirements;
– Standards describing general guidelines;
– Standards describing sector-specific guidelines.

Standards specifying requirements include the following:

– ISO/IEC 27001 “Information security management systems – Requirements” formally specifies ISMS against which thousands of organizations have been certified compliant;

– ISO/IEC 27006 “Requirements for bodies providing audit and certification of information security management systems” provides a

formal guidance for the for accredited organizations which certify other organizations compliant with ISO/IEC 27001;

– ISO/IEC 27009 “Sector-specific application of ISO/IEC 27001 – Requirements” at the time of 2016 is existing as a draft intended to provide guidance for those developing new ISO/IEC 27000 family standards.

Standards describing general guidelines include the following:

– ISO/IEC 27002 “Code of practice for information security controls” provides a reasonably comprehensive suite of information security control objectives and generally-accepted good practice security controls

– ISO/IEC 27003 “Information security management system implementation guidance” provides basic advices on implementing ISO/IEC 27001;

– ISO/IEC 27004 “Information security management – Measurement” provides description for a set of security metrics,

– ISO/IEC 27005 “Information security risk management” discusses risk management principles;

– ISO/IEC 27007 “Guidelines for information security management systems auditing” provides recommendations for auditing of management elements of the ISMS;

– ISO/IEC TR 27008 “Guidelines for auditors on information security management systems controls” provides recommendations for auditing the information security elements of the ISMS;

– ISO/IEC 27013 “Guidance on the integrated implementation of ISO/IEC 27001 and ISO/IEC 20000-1” combining ISO/IEC 27000 ISMS with ISO/IEC 20000 IT Service Management, particularly for ITIL (IT Infrastructure Library)

– ISO/IEC 27014 “Governance of information security” provide governing recommendations in the context of information security;

– ISO/IEC TR 27016 “Information security management – Organizational economics” provides economic theory applied to information security.

Standards describing sector-specific guidelines cover such domains as energy, medicine, telecommunications, finance, cloud computing and others.

For example, ISO/IEC 27010 “Information security management for inter-sector and inter-organisational communications” sharing

information on information security between industry sectors and/or nations, particularly those affecting “critical infrastructure”.

For more information concerning ISO/IEC 27000 see Part 9 of this multi-book.

1.3 Standards series ISO/IEC 15408

Standards series ISO/IEC 15408, which is also known as the Common Criteria includes the following three parts:

– ISO/IEC 15408-1 “Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model”;

– ISO/IEC 15408-2 “Information technology – Security techniques – Evaluation criteria for IT security – Part 2: Security functional components”;

– ISO/IEC 15408-3 “Information technology – Security techniques – Evaluation criteria for IT security – Part 2: Security assurance components”;

Part 1, “Introduction and general model” defines the general concepts and principles of IT security evaluation and presents a general model of evaluation (see Fig. 1.1). At the time evaluation concept is based on a confidence in correctness and sufficiency of security countermeasures (see Fig.1.2).

Part 2, “Security functional components” establishes a set of functional components that serve as standard templates upon which to base functional requirements for Targets of Evaluation (TOEs). ISO/IEC 15408-2 catalogues the set of functional components and organizes them in families and classes. There are the following classes of functional components described in ISO/IEC 15408-2: Security audit, Communication, Cryptographic support, User data protection, Identification and authentication, Security management, Privacy, Protection of the security functionality, Resource utilization, Access, Trusted path/channels.

Part 3, “Security assurance components” establishes a set of assurance components that serve as standard templates upon which to base assurance requirements for TOEs. ISO/IEC 15408-3 catalogues the set of assurance components and organizes them into families and classes. There are the following classes of assurance components

described in ISO/IEC 15408-3: Development, Guidance documents, Life-cycle support, Security Target evaluation, Tests, and Vulnerability assessment.

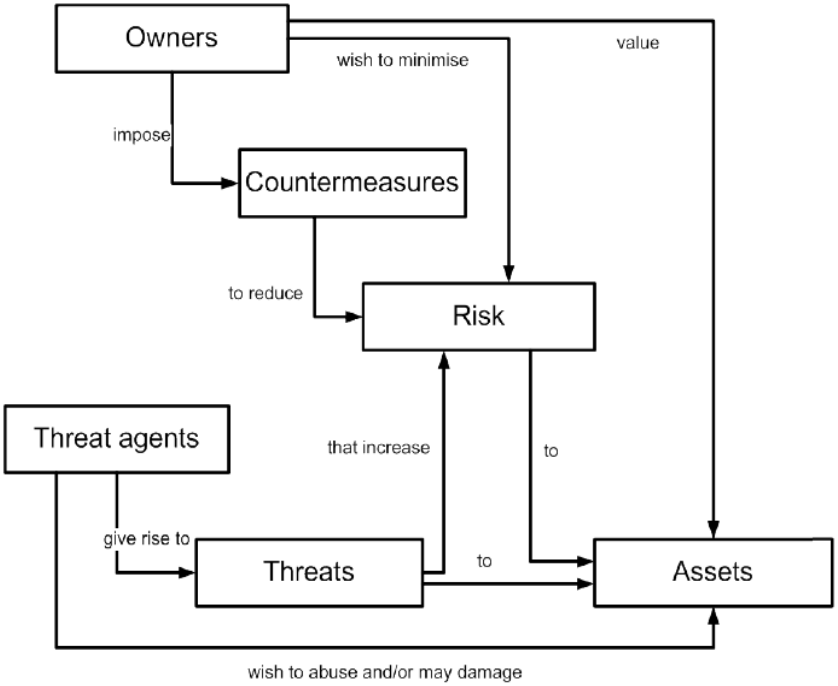


Fig. 1.1 – Security concepts and relationships
(source: ISO/IEC 15408-1)

ISO/IEC 15408-3 also defines evaluation criteria for Protection Profiles and Security Targets and presents seven pre-defined assurance packages which are called the Evaluation Assurance Levels (EALs). ISO/IEC 15408-3 states the following EALs:

– EAL1: functionally tested;
– EAL2: structurally tested;
– EAL3: methodically tested and checked;
– EAL4: methodically designed, tested, and reviewed;
– EAL5: semiformally designed and tested;

– EAL6: semiformally verified design and tested;
– EAL7: formally verified design and tested.

For more information concerning ISO/IEC 15408see Part 14 of this multi-book.

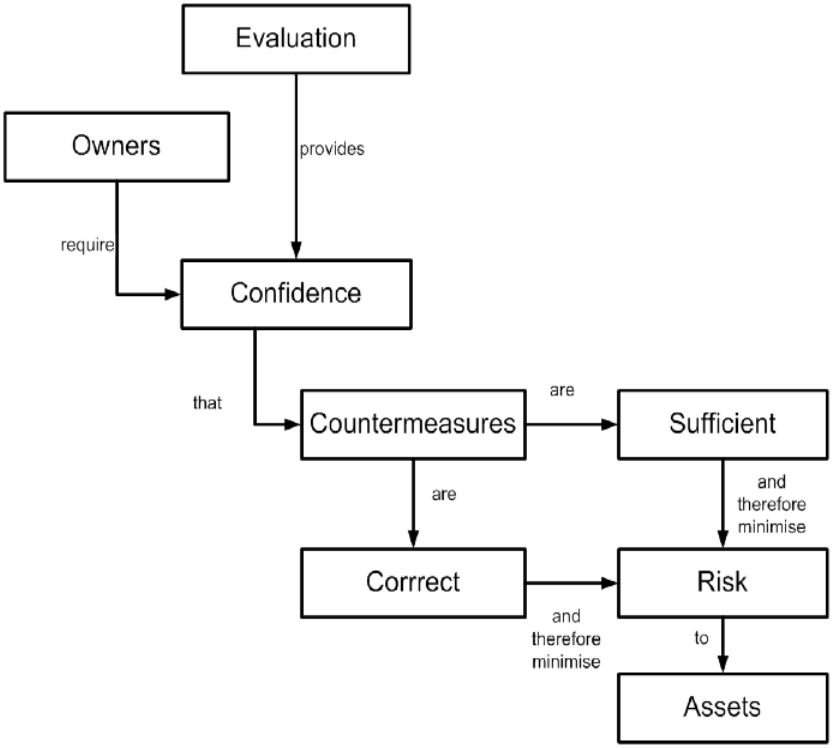


Fig. 1.2 – Evaluation concepts and relationships
(source: ISO/IEC 15408-1)

1.4 Standards series ISA/IEC 62443

Originally these standards have been developed by International Society of Automation (ISA) as series ANSI/ISA-99.00.

After that these standards have been adopted by International Electrotechnical Commission. At the present there are the following standards in force adopted by IEC:

– IEC TS 62443-1-1:2009 “Industrial communication networks – Network and system security – Part 1-1: Terminology, concepts and models”;

– IEC 62443-2-1:2010 “Industrial communication networks – Network and system security – Part 2-1: Establishing an industrial automation and control system security program”;

– IEC TR 62443-2-3:2015 “Security for industrial automation and control systems – Part 2-3: Patch management in the IACS environment”;

– IEC 62443-2-4:2015 “Security for industrial automation and control systems – Part 2-4: Security program requirements for IACS service providers”;

– IEC PAS 62443-3:2008 “Security for industrial process measurement and control – Network and system security”;

– IEC TR 62443-3-1:2009 “Industrial communication networks - Network and system security – Part 3-1: Security technologies for industrial automation and control systems”;

– IEC 62443-3-3:2013 “Industrial communication networks – Network and system security – Part 3-3: System security requirements and security levels”.

Now a structure of series is updated and new versions of the standards are in progress. ISA is developing master versions for the 62443 series, after that IEC should reissue identical standards. The developed 62443 series includes the following thirteen standards divided into four groups:

1) General:

– ISA/IEC 62443-1-1 “Terminology, concepts and models”;

– ISA/IEC 62443-1-2 “Master glossary of terms and abbreviations”;

– ISA/IEC 62443-1-3 “System security compliance metrics”;

– ISA/IEC 62443-1-4 “Industrial Automation and Control Systems (IACS) security lifecycle and use-case”;

2) Policies and Procedures:

– ISA/IEC 62443-2-1 “Requirements for an IACS security management system”;

– ISA/IEC 62443-2-2 “Implementation guidance for an IACS security management system”;

– ISA/IEC 62443-2-3 “Patch management in the IACS environment”;

– ISA/IEC 62443-2-4 “Installation and maintenance requirements for IACS suppliers”;

3) System:

– ISA/IEC TR 62443-3-1 “Security techniques for IACS”;

– ISA/IEC 62443-3-2 “Security levels for zones and conduits”;

– ISA/IEC 62443-3-3 “System security requirements and security levels”;

4) Component:

– ISA/IEC 62443-4-1 “Product Development Requirements”;

– ISA/IEC 62443-4-2 “Technical Security Requirements for IACS Components”.

The ISA/IEC 62443 series address the needs to design electronic security robustness and resilience into industrial automation control systems (IACS). Robustness provides the capabilities for the IACS to operate under a range of cyber-induced perturbations and disturbances. Resilience provides the capabilities to restore the IACS after unexpected and rare cyber-induced events. Robustness and resilience are not general properties of IACS but are relevant to specific classes of cyber -induced perturbations. An IACS that is resilient or robust to a certain type of cyber-induced perturbations may be brittle or fragile to another. Such a trade-off is the subject of profiles, which others can derive from the ISA/IEC 62443 requirements and guidelines. The goal in developing the ISA/IEC 62443 series is to improve the availability, integrity and confidentiality of components or systems used for industrial automation and control, and to provide criteria for procuring and implementing secure industrial automation and control systems. Application of the requirements and guidance in ISA/IEC 62443 is intended to improve electronic security and help to reducing the risk of compromising confidential information or causing degradation or failure of the equipment (hardware and software) of systems under control. The concept of IACS electronic security is applied in the broadest possible sense, encompassing all types of plants, facilities, and systems in all industries. Automation and control systems include, but are not limited to:

– Hardware and software systems such as DCS, PLC, SCADA, networked electronic sensing, and monitoring and diagnostic systems;

– Associated internal, human, network, or machine interfaces used to provide control, safety, and manufacturing operations functionality to continuous, batch, discrete, and other processes.

The requirements and guidance are directed towards those responsible for designing, implementing, or managing IACS. This information also applies to users, system integrators, security practitioners, and control systems manufacturers and vendors.

For more information concerning security assurance approach as it is described in ISA/IEC 62443, see Part 4 of this multi-book.

1.5 National Institute of Standards and Technology Cybersecurity Framework (NIST SCF)

NIST SP 800-53 “Security and Privacy Controls for Federal Information Systems and Organizations” provides a catalog of security controls measures. This catalog includes seventeen parts covering different organizational, technical and physical sides of security control (see Fig. 1.3).

Additionally NIST SP 800-53 it is a base for NIST CSF which harmonizes security control requirements with the following standards and good practices frameworks:

– ISO/IEC 27000 “Information security management systems” (see Section 1.2);

– ISA/IEC 62443 “Security for Industrial Automation and Control Systems”

– Control Objectives for Information and Related Technologies (COBIT) framework

– Center for Internet Security Critical Security Controls for Effective Cyber Defense framework (CIS CSC).

NIST CSF describes security activities by systematic way dividing into five the main functions: Identify, Protect, Detect, Respond, and Recover.

Each of the function is described through categories which include subcategories. Subcategories refer to Security Control Catalog (Appendix F of NIST SP 800-53), which provides a range of safeguards and countermeasures for organizations and information systems.

The following contains functions and categories description (see Fig. 1.4).

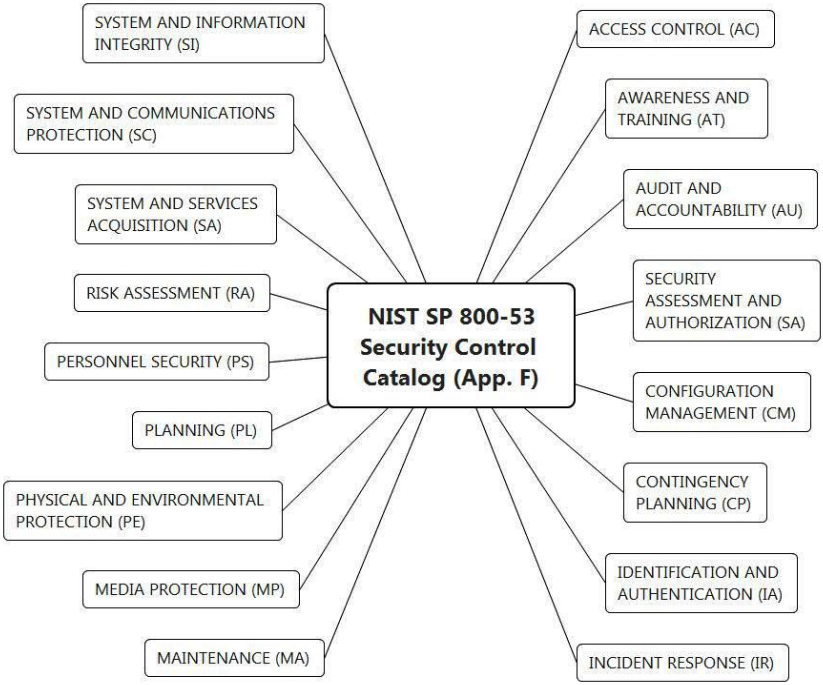


Fig. 1.3 – NIST SP 800-53: Structure of Security Control Catalog

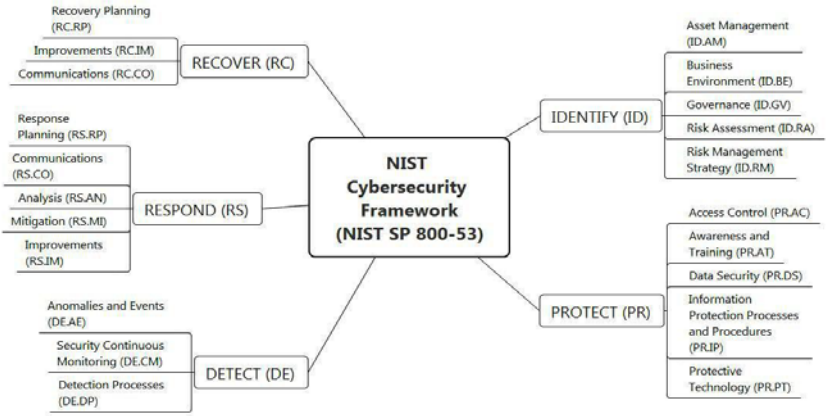


Fig. 1.4 – NIST SP 800-53: Cybersecurity Framework (NIST CCF)

"Identify" means to develop the organizational understanding to manage cybersecurity risk to systems, assets, data, and capabilities, what should be done with the following categories:

– Asset Management (ID.AM): The data, personnel, devices, systems, and facilities that enable the organization to achieve business purposes are identified and managed consistent with their relative importance to business objectives and the organization's risk strategy;

– Business Environment (ID.BE): The organization's mission, objectives, stakeholders, and activities are understood and prioritized; this information is used to inform cybersecurity roles, responsibilities, and risk management decisions;

– Governance (ID.GV): The policies, procedures, and processes to manage and monitor the organization's regulatory, legal, risk, environmental, and operational requirements are understood and inform the management of cybersecurity risk;

– Risk Assessment (ID.RA): The organization understands the cybersecurity risk to organizational operations (including mission, functions, image, or reputation), organizational assets, and individuals;

– Risk Management Strategy (ID.RM): The organization's priorities, constraints, risk tolerances, and assumptions are established and used to support operational risk decisions.

"Protect" means to develop and implement the appropriate safeguards to ensure delivery of critical infrastructure services, what should be done with the following categories:

– Access Control (PR.AC): Access to assets and associated facilities is limited to authorized users, processes, or devices, and to authorized activities and transactions;

– Awareness and Training (PR.AT): The organization's personnel and partners are provided cybersecurity awareness education and are adequately trained to perform their information security-related duties and responsibilities consistent with related policies, procedures, and agreements;

– Data Security (PR.DS): Information and records (data) are managed consistent with the organization's risk strategy to protect the confidentiality, integrity, and availability of information;

– Information Protection Processes and Procedures (PR.IP): Security policies (that address purpose, scope, roles, responsibilities, management commitment, and coordination among organizational

entities), processes, and procedures are maintained and used to manage protection of information systems and assets;

– Maintenance (PR.MA): Maintenance and repairs of industrial control and information system components is performed consistent with policies and procedures;

– Protective Technology (PR.PT): Technical security solutions are managed to ensure the security and resilience of systems and assets, consistent with related policies, procedures, and agreements.

"Detect" means to develop and implement the appropriate activities to identify the occurrence of a cybersecurity event, what should be done with the following categories:

– Anomalies and Events (DE.AE): Anomalous activity is detected in a timely manner and the potential impact of events is understood;

– Security Continuous Monitoring (DE.CM): The information system and assets are monitored at discrete intervals to identify cybersecurity events and verify the effectiveness of protective measures;

– Detection Processes (DE.DP): Detection processes and procedures are maintained and tested to ensure timely and adequate awareness of anomalous events.

"Respond" means to develop and implement the appropriate activities to take action regarding a detected cybersecurity event, what should be done with the following categories:

– Response Planning (RS.RP): Response processes and procedures are executed and maintained, to ensure timely response to detected cybersecurity events;

– Communications (RS.CO): Response activities are coordinated with internal and external stakeholders, as appropriate, to include external support from law enforcement agencies;

– Analysis (RS.AN): Analysis is conducted to ensure adequate response and support recovery activities;

– Mitigation (RS.MI): Activities are performed to prevent expansion of an event, mitigate its effects, and eradicate the incident;

– Improvements (RS.IM): Organizational response activities are improved by incorporating lessons learned from current and previous detection/response activities.

"Recover" means to develop and implement the appropriate activities to maintain plans for resilience and to restore any capabilities

or services that were impaired due to a cybersecurity event, what should be done with the following categories:

– Recovery Planning (RC.RP): Recovery processes and procedures are executed and maintained to ensure timely restoration of systems or assets affected by cybersecurity events;

– Improvements (RC.IM): Recovery planning and processes are improved by incorporating lessons learned into future activities;

– Communications (RC.CO): Restoration activities are coordinated with internal and external parties, such as coordinating centers, Internet Service Providers, owners of attacking systems, victims, and vendors.

Conclusions

There are a lot of dynamically developed standards in security domain.

Standards family ISO/IEC 27000 describes requirements to ISMS which are implemented in many countries. All ISO/IEC 27000 standards family can be divided in the three following sets:

– Standards specifying requirements;

– Standards describing general guidelines;

– Standards describing sector-specific guidelines.

However, many other standards and technical documents also endorse ISMS with diverse interpretations. NIST SP 800 53 [2], for example, is used in USA to establish and assess ISMS. NIST CSF is harmonized with ISO/IEC 27000, as well as with ISA/IEC 62443, COBIT, and CIS CSC. NIST CSF describes security activities by systematic way dividing into five the main functions: Identify, Protect, Detect, Respond, and Recover.

At the same time, ISMS is mainly managerial and organizational issue, like Quality Management System or Project Management. ISMS describes processes which should be organized with under a concept of “Plan – Do – Check – Act” cycle. It means that for IT systems another part of requirements should be applied. Such requirements should also cover:

– Risk Management and Assessment [7];

– Security Life Cycle [8];

– Security Levels [4];

– Failures and attack avoidance [9,10];

– Security and safety relation for critical systems [4].

Standards series ISO/IEC 15408 endorse Common Criteria for IT systems security assessment and provides security concepts including relations between basic security entities (risks, assets, threats, vulnerabilities, and countermeasures).

The most applicable requirements to Industrial Control Systems can be taken from NIST SP 800-82 [4] and ISA/IEC 62443 standards series.

Questions to self-checking

1. List a set of standards related with security issues.
2. List a set of organizations which develop security standards.
3. Which standards are more applicable for security of Industrial Control Systems (ICS)?
4. Which standards are more applicable for security of web-systems?
5. Which standards are more applicable for security of Internet of Thing (IoT)?
6. Which main issues are covered in security standards?
7. Describe structure of ISO/IEC 27000 standards family.
8. Describe structure of ISO/IEC 15408 standards series.
9. What is a security concept stated in ISO/IEC 15408 standards series?
10. Describe structure of ISA/IEC 62443 standards series.
11. Describe structure of NIST Cybersecurity Framework.
12. Which are the main issues of Security Management System?

References

1. T. Nguyen, T. Levin, C. Irvine. High robustness requirements in a Common Criteria protection profile // Proceeding of 2006 IEEE 4th International Workshop on Information Assurance (IWIA). – P.78-87.

2. NIST SP 800-53 Revision 4, Security and Privacy Controls for Federal Information Systems and Organizations. – National Institute of Standards and Technologies, 2015. – 462 p.

3. NIST SP 800-53A Revision 4, Assessing Security and Privacy Controls in Federal Information Systems and Organizations:

Building Effective Security Assessment Plans. – National Institute of Standards and Technologies, 2014. – 487 p.

4. NIST SP 800-82 Revision 2, Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations such as Programmable Logic Controllers (PLC). – National Institute of Standards and Technologies, 2015. – 247 p.

5. AGA Report No. 12, Cryptographic Protection of SCADA Communications, Part 1: Background, Policies and Test Plan. – American Gas Association, 2006. – 123 p.

6. Common Cybersecurity Vulnerabilities in Industrial Control Systems. – U.S. Department of Homeland Security, 2011. – 76 p.

7. NIST SP 800-39, Managing Information Security Risk: Organization, Mission, and Information System View. – National Institute of Standards and Technologies, 2011. – 88 p.

8. Nuclear Power Plant Instrumentation and Control Systems for Safety and Security / Yastrebenetsky M., Kharchenko V. (Edits). – IGI Global. – 2014. – 470 p.

9. O. Netkachov, P. Popov, K. Salako. Model-Based Evaluation of the Resilience of Critical Infrastructures Under Cyber Attacks // Proceeding of 9th International Conference (CRITIS 2014). – P. 231-243.

10. S. Srinivasan, R. Kumar, J. Vain. Integration of IEC 61850 and OPC UA for Smart Grid automation // 2013 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia). – P. 1-5.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ К РАЗДЕЛУ 2

AGA – American Gas Association

C2M2 – Cybersecurity Capability Maturity Model

CIS – Center for Internet Security

CIS CSC – CIS Critical Security Controls for Effective Cyber Defense (framework)

COBIT – Control Objectives for Information and Related Technologies (framework)

DHS – the U.S. Department of Homeland Security

DOE – the U.S. Department of Energy

EAL – Evaluation Assurance Level

ES-C2M2 – Electricity Subsector Cybersecurity Capability Maturity Model

IEC – International Electrotechnical Commission

IEEE – Institute of Electrical and Electronics Engineers

ICS – Industrial Control System

ICS-CERT – Industrial Control Systems Cyber Emergency Response Team

ISMS – Information Security Management System

ISA – International Society of Automation

ISO – International Standardization Organization

FISMA – Federal Information Security Management Act

NERC – North American Electric Reliability Corporation

NIST – National Institute of Standards and Technology

NIST SCF – NIST Cybersecurity Framework

NIST SP – NIST Special Publication

OWASP – Open Web Application Security Project

R&D – Research and Development

SCADA – Supervisory Control And Data Acquisition

TOE – Targets of Evaluation

АННОТАЦИЯ

В разделе рассмотрены стандарты в области информационной безопасности. Приведен перечень основных существующих на данный момент стандартов. Дана характеристика наиболее важных стандартов (ISO/IEC 27000, ISO/IEC 15408, ISA/IEC 62443, NIST SP 800-53).

У розділі розглянуто стандарти у галузі інформаційної безпеки. Наведено перелік основних існуючий у дійсний момент стандартів. Дана характеристика найбільш важливих стандартів (ISO/IEC 27000, ISO/IEC 15408, ISA/IEC 62443, NIST SP 800-53).

Information security standards are discussed in the section. List of the main actual security standards is given. Contents of the most important security standards (ISO/IEC 27000, ISO/IEC 15408, ISA/IEC 62443, NIST SP 800-53) are considered.

2 BASICS OF CRYPTOLOGY FOR RESILIENT COMPUTING

2.1 Introduction

Cryptology take a special place in security. This science involves studies in two main directions: cryptography and cryptanalysis. The core of cryptography is secure communication. The security should guarantee that eavesdropper, who observes the text sent across the channel, could figure out nothing about message. For ages cryptography has been used to provide the secrecy of mostly military or diplomatic communications. Due to the growth of electronic commerce and the Internet itself the notation of secure communication is much more wider nowadays and includes protocols of web traffic (SSL, TLS), wireless traffic (WEP, WPA, WPA2), cell-phone-traffic (GSM) and so on.

Initially cryptography was considered only as a tool to ensure confidentiality. Confidentiality is the term used to describe the prevention of accessing the information by unauthorized computers or users. Today, cryptography has a much wider reach, covering not only confidentiality of communications and stored data, but also guaranteeing identity, integrity, entity authentication, and data origin authentication and provenance etc. Practical applications of cryptography includes content protection, digital signatures, anonymous communication, e-voting, zero-knowledge proofs etc.

While cryptography is concentrated on construction of secure cryptosystems, cryptanalysis goal is to reveal information from hidden messages sent over an insecure channel without secret knowledge. It is also known as code cracking. Usually the security of a cryptosystem is proven for an abstract mathematical algorithm in a formal model of computation under certain types of attacks. However all practical cryptosystems are actually semantically secure. It means that adversaries who have sufficient amount of time and resources can crack almost any algorithm and access encrypted information. A more realistic goal of cryptography is to make breaking a cryptosystem complicated and time-consuming task for an attacker, restricted by limited resources. This, however, is not the end of a story: the security

must hold for the actual implementation of the algorithm in the real world in which this algorithm is run. A crucial difference about the two scenarios is that in the former we assume that secret keys are indeed secret, and the adversary has no information about them – our proofs crucially rely on this fact.

In reality cryptographic algorithms are routinely run in adversarial settings, where keys could be compromised and the adversary might gain some information about those secrets, by observing the behavior of the algorithm, in ways not captured by our formal computational model. Side-channel attacks exploit the fact that computing devices leak information to the outside world not just through input-output interaction, but through physical characteristics of computation such as power consumption, timing, and electro-magnetic radiation. Such information leakage betrays information about the secrets during cryptosystem execution, which cannot be efficiently derived from access to mathematical object alone. Physical attacks have been successfully utilized to break many cryptographic algorithms in common use. Attacks such as these have broken systems with a mathematical security proof, without violating any of the underlying mathematical principles. Physical leakages are particularly accessible when the device is at the hands of an adversary, as is often the case for modern devices such as smart-cards, TPM chips, mobile phones and laptops.

Leakage-resilient (or side-channel resilient) cryptography attempts to tackle such attacks. One main goal is building more robust models of adversarial access to a cryptographic algorithm, and developing methods grounded in modern cryptography to provably resist such attacks.

Although building efficient cryptosystem resilient to physical leakages and tampering people needs understanding the fundamentals of modern crypto-primitives.

2.2 Terminology. Classification of cryptosystems.

Let clarify the terminology.

Plaintext (message) is ordinary readable text before being encrypted

$$"m_1 m_2 m_3 ... m_l"$$

where $m_j \in A_n,\ j = 1..l,$ A_n is an alphabet of n characters.

Alphabet is a finite set of characters, which are used for information coding. For instance, A_{26} could be a set of letters of English alphabet; A_{256} could be considered a set of symbols from ASCII table; A_2={0,1} is a binary alphabet.

*Ciphertext (*cyphertext*) is encrypted plaintext:*

$$c_1 c_2 c_3 ... c_l",$$

where $c_j \in A_n,\ j = 1..l.$

Secret key is a parameter that is used to encrypt and decrypt messages:

$$k_2 k_3 ... k_l", k_j \in A_n,\ j = 1..l.$$

Encryption is a process of conversion of information (plaintext) into another form (ciphertext), which cannot be easily understood by anyone except owner of secret key.

Decryption is the reverse process to encryption conversion of ciphertext into plaintext with the secret key.

The simplest encryption methods date back to 2000-3000 BC when the ancient Greeks and Romans sent secret messages by substituting or permutation letters. There is a sufficiently large number of transposition ciphers. Among them are a Scytale cipher, anagrams, and variety of so called route ciphers: rail fence, columnar transposition, double transposition, Myszkowski transposition, Cardan grille. The main idea of all of them is change the location of symbols in plaintext according to some predefined permutation rules. There were much more substitution ciphers, which are based on substitution of plaintext symbol by symbol of ciphertext. They could be subdivided on two large groups: monoalphabetic ciphers (Polybius square, Caesar cipher, affine cipher, Trithemius cipher) and polyalphabetic ciphers (bigram affine cipher, Playfair cipher, hill cipher, Vigenère cipher). The difference is that in monoalphabetic ciphers fixed symbol of plaintext is substituted with the same symbol of ciphertext. In polyalphabetic ciphers same symbol can be replaced with different symbols depends on its position in the plaintext.

Symmetric encryption was the only type of encryption in use from the ancient till 1970s. Symmetric encryption transforms plaintext into ciphertext using a secret key and an encryption algorithm. Using the same key and a decryption algorithm, the plaintext is recovered from

the ciphertext. Development of computers and Internet causes drastic changes in symmetric cryptography: classical methods were replaced with modern digital approaches to encryption: block and stream ciphers. Stream ciphers encrypt each binary digit in a data stream individually. This is usually achieved by adding a bit from a key stream to a plaintext bit. So the main problem to solve is the generation of the key sequence with good statistical properties. Block ciphers processes the blocks of plaintext, which has to be of fixed length. It means that plaintext need to be divided on blocks in advance. Both block and stream ciphers have some advantages, which we will consider later on. Nevertheless all symmetric encryption methods have one serious drawback, what is key management and distribution problem. This difficulty can be overcome using asymmetric cryptography. Unlike traditional cryptographic methods quantum cryptography is based on physics, not mathematics. The best known example of quantum cryptography is quantum key distribution which offers an information-theoretically secure solution to the key exchange problem. Currently used popular public-key encryption and signature schemes can be broken by quantum adversaries

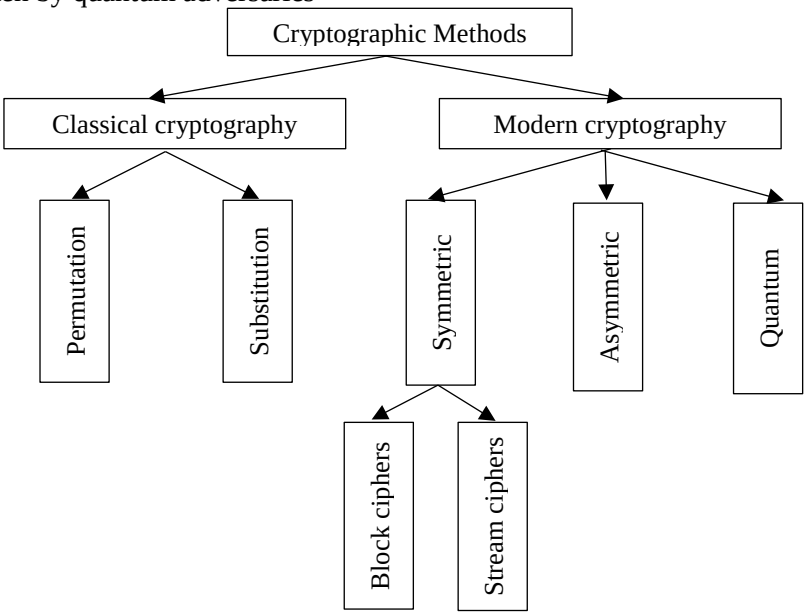


Fig.2.1 Classification of cryptographic methods

Which cryptographic methods are the best? Does there exist a perfect cipher? Let clarify these questions further.

2.3 Perfect and computational secrecy

There is a critical difference between the decryption performed by the legitimate user and cryptanalysis performed by unauthorized person. Features and capabilities of potential attacker determine the requirements for reliable encryption. One of the key steps in the development of the secrecy model of cryptosystem is to define threat model and security goal.

Recall that the main goal of cryptanalysis is to restore the plaintext without knowledge of the key or to recover the secret key.

As a basic starting point it is normally assumed that, for the purposes of cryptanalysis, the general algorithm is known (Kerckhoffs' principle).

Attacks can be classified based on what type of information the attacker has available. Table 1 includes the most common threat models for encryption.

Table 2.1 – Types of cryptographic attacks

	Type of attack	**Information known to cryptanalytist**
1	Ciphertext Only Attack (COA)	Encryption algorithm; One or more ciphertexts c_i Brute Force Attack (attacker tries all possible keys) can be applied in this case. Modern cryptosystems are guarded against ciphertext-only attacks.
2	Known Plaintext Attack (KPA)	Encryption algorithm; One or more plaintext–ciphertext pairs (m_i-c_i), formed with the secret key The best example of this attack is linear cryptanalysis against block ciphers.
3	Chosen Plaintext Attack (CPA)	Encryption algorithm; One or more plaintext–ciphertext pairs (m_i-c_i), formed with the secret key, but unlike previous attack plaintext messages are chosen by cryptanalyst.

		An example of this attack is differential cryptanalysis applied against block ciphers as well as hash functions. A popular public key cryptosystem, RSA is also vulnerable to chosen-plaintext attacks.
4	Chosen Ciphertext Attack (CCA)	Encryption algorithm; One or more plaintext–ciphertext pairs (m_i.c_i), formed with the secret key, but unlike previous attack ciphertexts could be chosen by cryptanalyst Ciphertexts are chosen in advance (*lunchtime attack)*
5	Adaptive Chosen Plaintext and Chosen Ciphertext Attacks	Encryption algorithm; One or more plaintext–ciphertext pairs (m_i.c_i), formed with the secret key but unlike attacks 3 and 4 adversary subsequent plain- or ciphertexts based on information learned from previous encryptions
6	Related-key attack	Like a chosen-plaintext attack, except the attacker can obtain ciphertexts encrypted under two different keys. The keys are unknown, but the relationship between them is known; for example, two keys that differ in the one bit
7	Side-Channel Attacks	These attacks are launched to exploit the weakness in physical implementation of the cryptosystem. Some additional information is known to attacker, for instance computation time, power consumption, leaked electromagnetic radiation and so on. First implementation of AES were vulnerable to such attacks.

Cryptosystem can be protected from one type of attacks and be vulnerable towards others. Formally, secrecy is understood as ability of cryptosystem remained resistant to cryptographic attacks. Leakage Resilient Cryptography tries to provide provably secure primitives in the presence of a wide range of side-channel information.

K. Shannon first introduced the concept of secrecy of cryptosystem and took into account Ciphertext Only Attack (threat model). Attacker was assumed to have unlimited computing resources.

In his book "Communication Theory of Secrecy Systems" Shannon considered so-called symmetric system, those in which encryption and decryption uses the same key.

A symmetric (private-key) cryptosystem defined over *(M,K,C)* can be described in mathematical terms as a pair of "efficient" algorithms *(Enc,Dec)*, such that:

– Enc: $M \times K \rightarrow C$ – (encryption algorithm): takes key $k \epsilon K$ and message $m \epsilon M$ as inputs; outputs ciphertext $c = Enc(k,m)$.

– Dec: $C \times K \rightarrow M$ – (decryption algorithm): takes key $k \epsilon K$ and ciphertext $c \epsilon C$ as input; outputs message $m = Dec(k,c)$, such that $\forall k \in K$ and $m \in M$ it is valid that $Enc(k,Dec(k,m)) = m$ (consistency equation).

Here:

M – message space;

K– key space;

C – ciphertext space.

Algorithm *Enc* could often a randomized algorithm. On the other hand the decrypting algorithm *Dec* is always deterministic

Often the pair (Enc, Dec) can be supplemented with one more algorithm Gen, which generates extended key from small key-seed.

In general the structure of symmetric cryptosystem is illustrated on Fig.2.2.

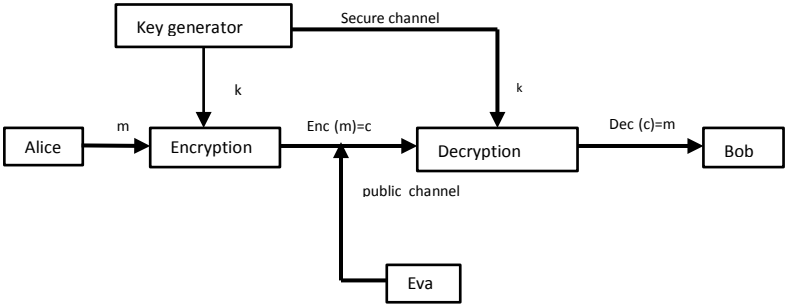


Fig.2.2 Scheme of symmetric system

We have two parties, Bob and Alice, who have shared a secret key k across the secure channel in advance. k-value could be generated using a particular deterministic algorithm. When Alice has some message m that she wants to send to Bob she will encrypt that message, using the encryption algorithm and their shared key k. This results in a ciphertext that Alice sends across the public channel to Bob. Upon receiving this message, Bob will use his key to decrypt the ciphertext

and recover the original message. At a high level, both parties are trying to ensure secrecy of their communication against an eavesdropper Eva who can observe everything being sent across the public channel between Alice and Bob.

Shannon believed that attacker would not be interested only in getting the entire secret message or key but also in some additional information about plaintext. The system has perfect secrecy by Shannon if regardless of any prior info the attacker has about the plaintext, the ciphertext should leak no additional information about the plaintext.

Besides the assumption that Ciphertext Only Attack is only possible attack and only one ciphertext is available scientist assumed that knows the probability distributions of messages *P(M)* and keys *P(K)*.

Encryption scheme *(Enc, Dec)* with message space *M*, key space *K* and ciphertext space *C* is perfectly secret if for every distribution over *M*, every $m \epsilon M$, and every $c \epsilon C$ with P(c)>0, it holds that

$$P(m \mid c)=P(m)$$

So perfect secrecy means that observing the ciphertext should not change the attacker's knowledge about the distribution of the plaintext.

Equivalent definition of perfect security could be formulated in terms of entropy.

For $\forall m \in M, c \in C$ $\quad H(m \mid c)=H(m)$

It can treated as follows: attacker gets zero information from ciphertext about plaintext $I = H(m) - H(m|c) = 0$.

Shannon also proved the validity of few lemmas, which are based on definition of perfect secrecy.

Lemma 2.1.

Cryptosystem has perfect secrecy if $\forall m \in M, c \in C$ the equilaty:

$$P(c \mid m)=P(c)$$

is valid.

Lemma 2.2.

Cryptosystem with perfect secrecy will satisfy the inequality

$$\#K \geq \#C \geq \#M$$

This lemma is carrying bad news, because the total number of keys has to be at least not less the number of messages. Shannon also gave the instructions how to construct the ideal system in next lemma.

Lemma 2.3.

If ⟨M,C,K, Enc(k,·),Dec(k,·)⟩ describes a particular symmetric cryptosystem and #K=#C=#M, it will have perfect secrecy only and only if

- The distribution of keys is uniform $P(k)=1/(\#K)$ for $\forall k \in K$
- There is only one $k \in K$ for each pair $m \in M$, $c \in C$ such that $Enc(k,m)=c$.

The question is: Does the ideal system exist? It turns out the answer is "Yes"

Gilbert Vernam invented and patented his cipher called also One-time pad (OTP) in 1917. Let consider some details of algorithm

Message space $M=\{0,1\}^n$ is a set of all possible n-length bit strings. Key is selected randomly on uniformly distributed set $K = \{0,1\}^n$.

Encryption: $c_i=m_i \oplus k_i$ ($\oplus$ is a bit-wise XOR)

Decryption: $m_i=c_i \oplus k_i$

Illustration of OTP encryption scheme is depicted in Fig.2.3.

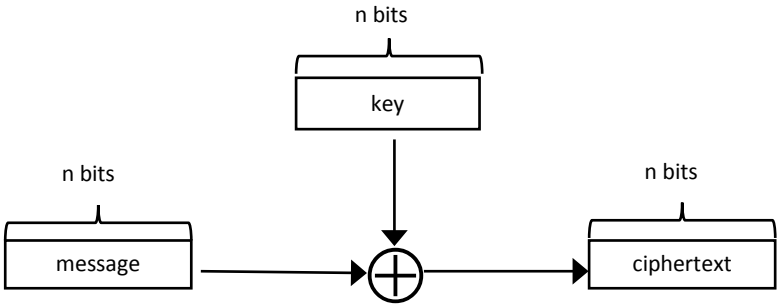


Fig.2.3 The encryption algorithm of OTP

Shannon has also proven the following lemma.

Lemma 2.4. Vernam cipher has perfect secrecy.

Despite of the fact that OTP cipher is ideal it is hard to use it in practice.

Drawbacks of the practical usage of the OTP cipher:

- Key has to be as long as message.
- Key has to used only once

Problem arisen if key is used more than once

- In worst case chosen plaintext attack $k = m \oplus c$

– Otherwise ciphertext will give us information about plaintext as $c_1 \oplus c_2 = m_1 \oplus m_2$.

Computational secrecy

In particular, in real life people are using encryption schemes with keys shorter than the message size to encrypt all sort of important information including credit card numbers. Could we use the proof of the impossibility result to break these schemes? Most cryptographic methods we use now are computationally secure. There is no strict mathematical definition of computational secrecy (semantic security). The following is true about computational secrecy:

1. Computational secrecy allows an attacker to learn information about the message with small probability.

2. Computational secrecy currently relies on unproven assumptions.

3. Computational secrecy only ensures secrecy against attackers running in some bounded amount of time or restricted computational resources.

2.4 Stream ciphers

Since the Vernam cipher is unconditionally secure but not very practical, it is natural that people would like to come up with the scheme, which uses shorter key. Although this statement refuses the necessary condition of perfect secrecy OTP, according to which key has to be as long as message. The core ideas of modern stream ciphers are:

– the encryption and decryption algorithms remain the same to OTP cipher;
– to replace “random” key by “pseudorandom” key.

So, the only difference between stream and OTP ciphers is that key is generated by deterministic algorithm from shorter secret key called seed. That is why the main problem to be solved while implementing stream ciphers is pseudo-random sequence modeling (γ-sequences).

The general scheme of any stream cipher is given in Fig.2.4

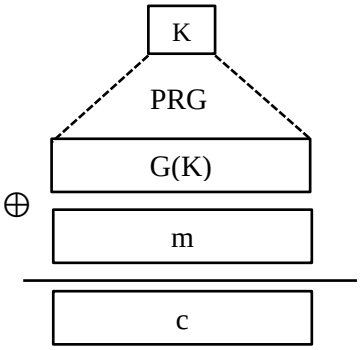


Fig.2.4 Illustration of stream cipher encryption algorithm

Operation ⊕ usually means bit-wise XOR. A pseudo-random generator (PRG) is an efficient, deterministic algorithm that expands a short, uniform seed into a longer, pseudorandom sequence. We would like this sequence to be random, but with a finite state machine and a deterministic algorithm we can not get a real randomness. Moreover, PRG will always generate a sequence, which is ultimately periodic. American standard NIST uses 15 tests to qualify pseudo-random sequence.

One of the important features of good PRG is unpredictability. There are the some PRGs which can be used in other areas but are weak for cryptography because they are predictable. Let consider one example of such weak PRG - linear congruential generator.

Linear congruential generators are generators defined like:

$$x_j = ax_{j-1} + b \bmod n$$

Variables *a*, *b* and *n* are constants. Value x_0 is supposed to be a seed. Period of such generator is not larger than *n*. These generators are very fast but unfortunately can not be used in cryptography.

Linear Feedback Shift Register (LFSR)

Linear feedback shift registers are useful tools in both coding theory (error checking and correction) in cryptography (generation of pseudo-random numbers). LFSRs are very fast PRGs and need very little hardware. Generating the pseudo-random numbers only requires a right-shift operation and an XOR operation. They have nice statistical properties and a well developed theory.

In fact, LFSR contains memory cells or stages each holding one bit of information. The content of cells is referred to as state of register. Each time the contents of several predefined cells are fed to the input of feedback function. It would be reasonable to use non-linear function as feedback. However, it is difficult to implement, that is why linear feedback function is used in practice. The most commonly used linear function of single bits as was said earlier is exclusive-or (XOR).

Fig.2.5 illustrates how LFSR works.

Both feedback coefficients $a_1 a_2,\ldots a_{n-1},a_n$ and values of register state are elements of finite field GF(2). Number n is called length of LFSR. The values $s_{n-1}, s_{n-2}, s_{n-3}, \ldots, s_1, s_0$ initially loaded into register specify the initial state. The initial state of register actually represents a seed and can be chosen arbitrary.

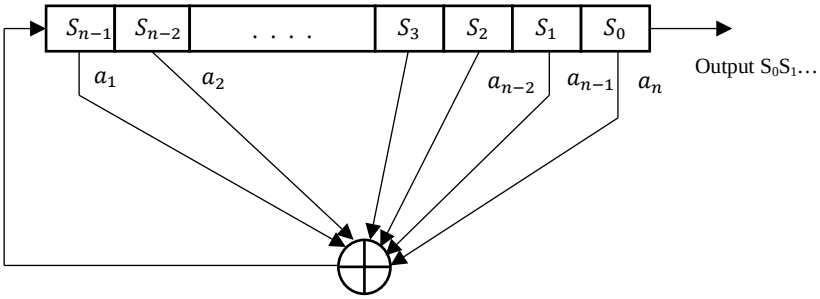


Fig.2.5 A general Linear Feedback Shift Register

Register works in discrete time moments in such a way that:

- Output the most right bit of the register s_0.
- Shift the content of s_i to the cell s_{i-1}, $i = \overline{1, n-1}$
- New value of the most left cell is the feedback bit computed as exclusive OR of values $s_{n-1}, s_{n-2}, s_{n-3}, \ldots, s_1, s_0$ multiplied by $a_1 a_2,\ldots a_{n-1},a_n$

If some coefficients $a_1 a_2,\ldots a_{n-1},a_n$ equals to zero then correspondent taps are removed from the scheme. The set of the taps with coefficients “one” is called tap sequence.

Mathematically the sequence s_i $i = \overline{1, N}$ generated by a shift register is just a sequence satisfying the n-term recursion

$$s_{n+t}=a_1 s_{n+t-1}\oplus a_2 s_{n+t-2}\oplus\ldots\oplus a_{n-2} s_{t+2}\oplus a_{n-1} s_{t+1}\oplus a_n s_t=\sum_{j=1}^{n} a_j\, s_{n+t-j}$$

The last formula describes the feedback function. It is called the recursion law which generate the sequence.

Output sequence of LFSR can be uniquely defined by feedback polynomial and initial state of register.

LFSR of length *n* with coefficients $a_1 a_2,\ldots a_{n-1},a_n$ has feedback polynomial:

$$P(x)=1\oplus\sum_{j=1}^{n} a_i\, x^i=x^n\oplus a_1 x\oplus a_2 x^2\oplus\ldots\oplus a_{n-1}x^{n-1}\oplus a_n x^n$$

Alternatively register output can be defined by characteristic polynomial of LFSR.

$$P^*(x)=x^n\oplus\sum_{j=1}^{n} a_j x^{n-j}=x^n\oplus a_1 x^{n-1}\oplus a_2 x^{n-2}\oplus\ldots\oplus a_{n-1}x\oplus a_n$$

The polynomial degree is defined by register length.

LFSR shown in Fig.2.6 has feedback polynomial $P(x)=1+x^3+x^4$, characteristic polynomial $P^*(x)=1+x+x^4$, recursion law $s_{4+t}=s_{t+1}+s_t$.

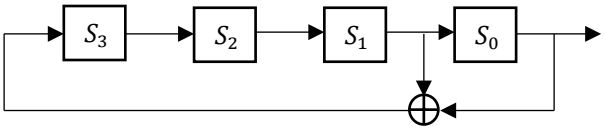


Fig.2.6 The example of LFRS

The register states in different time moments are given in following table:

	S_3	S_2	S_1	S_0
t=0	1	1	0	1
t=1	1	1	1	0
t=2	1	1	1	1
t=3	0	1	1	1
t=4	0	0	1	1
t=5	0	0	1	1

The output sequence is 1011110001001101

The LFRS sequence with maximal period is called m-sequence. Sequences of maximal period are of special interest. They can be produces only by specific polynomials called primitive.

LFSR have long been used as pseudo-random number generators for use in stream ciphers. Note that stream ciphers need pseudorandom sequences with a very large period. It is not trivial task to find a primitive polynomial with sufficiently large degree. Moreover, a stream cipher based on one LFSR which is a linear system is vulnerable to certain attacks.

Three general methods are employed to reduce this problem in LFSR-based stream ciphers:

- A filter generator composed of single LFSR whose output is a non-linear combination of several bits from the LFSR state;
- A combination generator is composed of several LFSRs whose outputs are combined by non-linear boolean function;
- LFSRs with irregular clocking (Stop and Go generator or Step1-Step2 generator). The keystream is produced by one or several LFSRs, but some LFSR bits decide which LFSR to clock and how often.

Important LFSR-based stream ciphers include A5/1 and A5/2, used in GSM cell phones.

A5 stream cipher

A5/1 is used in most European countries, whereas a weaker cipher, called A5/2, is used in other countries. There also exists A5/3 modification approved for 3G networks

Let describe the first of all A5 algorithms - an algorithm A5/1.

The description of A5/1 was first kept secret but its design has been finally published in Internet.

A5 contains of three LFRS with lengths 19, 22 i 23, which described by following feedback polynomial:

$$P_1(x)=x^{19}+x^{18}+x^{17}+x^{14}+1$$
$$P_2(x)=x^{22}+x^{21}+1$$
$$P_3(x)=x^{23}+x^{22}+x^{21}+x^{8}+1$$

The output is the XOR of all three registers.

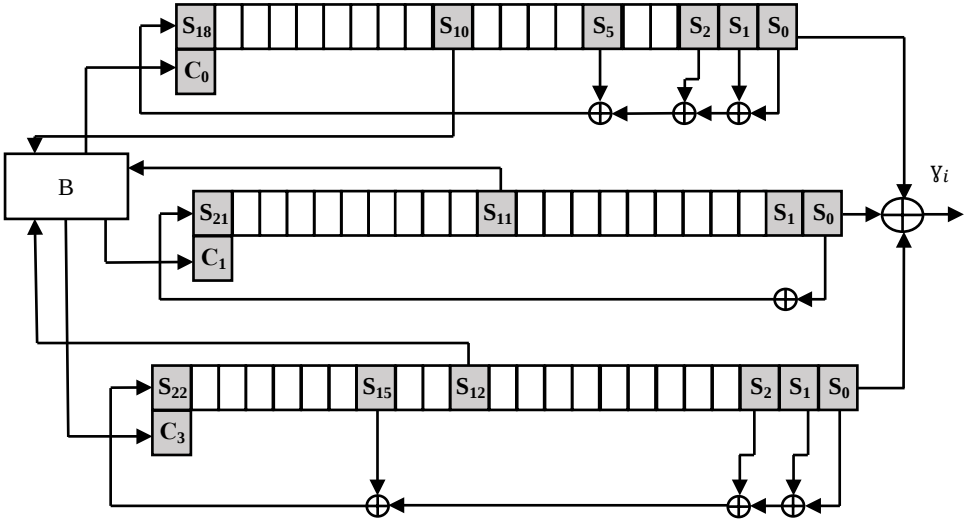


Fig.2.7 The structure of A5/1

For each frame transmission, the three LFSRs are first initialized to zero. Then, at time *t= 1,...,64*, the LFSRs are clocked, and the secret key bit K_t is XORed to the feedback bit of each LFSR. For t= *65,...,86*, the LFSRs are clocked in the same fashion, but the *(t-64)-th* bit of the frame number is now XORed to the feedback bits.

After these 86 cycles, the generator runs as follows (by principle Stop and Go):

- Each register has a clocking tap $\tau_1=s_{10}$ (LFSR1), $\tau_2=s_{11}$(LFSR2), $\tau_3=s_{12}$(LFSR3), which are inputs for block B.
- At each unit of time, the majority value of the 3 clocking bits is computed. A LFSR is clocked if and only if its clocking bit is equal to $F=\tau_1\wedge\tau_2\oplus\tau_1\wedge\tau_3\oplus\tau_2\wedge\tau_3$. Minimum two registers are working in each moment.
  - If $\tau_i = \tau_j \neq \tau_k \quad 1 \leq i,j,k \leq 3$, then $c_i = c_j = 1, c_k = 0$
  - If $\tau_i = \tau_j = \tau_k \quad 1 \leq i,j,k \leq 3$, then $c_i = c_j = c_k = 1$
- Output bit of keystream is the XOR output registers bits.

Famous cryptanalyst at Cambridge Ian Cassells said "Cryptography is a mixture of mathematics and muddle, and without the muddle mathematics can be used against you" He meant that to study and to proof the secrecy of ciphers we need to base on strong mathematical structures. At the same time to make ciphers stronger to attacks we need to add nonlinear muddle. This is true with respect to both stream and block ciphers.

2.5 Block ciphers

Stream ciphers are much faster than their closest competitors - block ciphers but only in the case if stream encryption is implemented in hardware. Block ciphers are easier to implement in software, since we can avoid significant manipulation of bits and operate data blocks which are more convenient for computers.

DES (Data Encryption Standard)

In 1974 the National Bureau of Standards (NBS) solicited the American industry to develop a cryptosystem that could be used as a standard in unclassified U.S. Government applications. IBM developed a system called LUCIFER. NBS involved National Security Agency (NSA) to review and assess this cipher. After being modified and simplified during review time, this system became the Data Encryption Standard (DES) in 1977.

DES parameters:

– Length of plaintext and ciphertext block is 64 bits.

– Key length is 64 bits, but the least significant bit of each key byte is used for parity check and could be ignored. Effective keysize is 56 bits

– Number of rounds is 16.

DES is an example of a Feistel cipher. Horst Feistel was one of the inventors of cipher LUCIFER – DES predecessor. A Feistel network works by splitting the data block into two equal pieces and applying encryption in multiple rounds. Each round implements permutation and combinations derived from the primary function or key. The number of rounds varies for each cipher that implements a Feistel network.

The structure of DES is given in Fig.3.8.

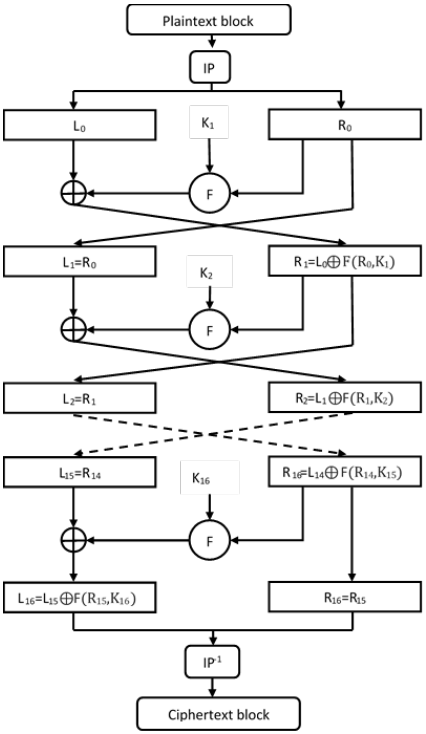


Fig.2.8 The general structure of DES

IP stands for an initial permutation, IP^{-1} stands for final permutation which is inverse to initial one. The initial and final permutations are straight permutation boxes, illustrated in Fig.2.9.

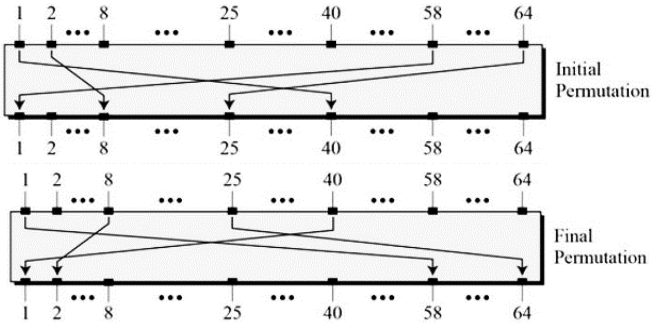


Fig.2.9 How IP and IP^{-1} work

IP and IP^{-1} have no cryptographic significance in DES

The block of 64 input bits is divided into two halves: the 32 leftmost bits form L_0 and the 32 rightmost bits form R_0. DES consists of 16 identical rounds. In each round, new contents of L_i and R_i are defined by formula

$$L_i = R_{i-1} \quad R_i = L_{i-1} \oplus F(K_i, R_{i-1})$$

K_i stands for round key, derived from main secret key K.

The heart of this cipher is the DES pseudo random function F. The DES function takes 48-bit key and 32 rightmost bits to produce a 32-bit output (see Fig.2.10 for details).

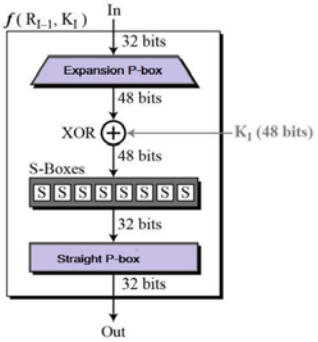
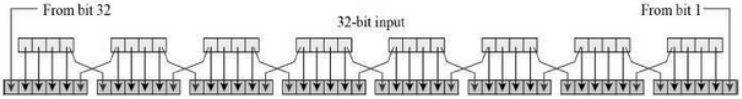
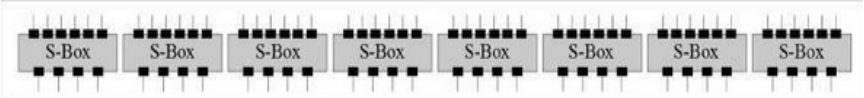


Fig.2.10 DES round function

Expansion Permutation Box − Since right input is 32-bit, we first need to expand right input to 48 bits to XOR then with 48-bit round key. Permutation logic is graphically depicted in the following illustration.



Substitution Boxes − The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –



S-blocks are provided in a form of lookup tables of size 4x16. The numbers from 0 to 15 are written in each row in some specified order. Six input bits define the number of row (the first and sixth bits) and column (four middle bits). The output is binary representation of number which stands in intersection of predefined row and column.

Straight Permutation Box – The 32 outputs from the S-boxes are rearranged according to a fixed permutation, often called the P-box.

The last but not the least thing left is to describe the procedure of round key generation.

Round key schedule

DES uses its key schedule in this way. Initially, 56 significant bits of the key are selected from the initial 64 and permuted. Then 56-bit key is split into two parts. In successive rounds, both halves are cyclically shifted left by one or two bits (specified for each round), and then 48 subkey bits are selected by final Permuted Choice.

A cryptographic system based on Feistel cipher structure uses the same algorithm for both encryption and decryption. Feistel network is a design model from which many different block ciphers are derived, not only DES.

Many people have criticized the decision to make DES a standard. The two main objections were:

– The effective keysize (56 bits) is too small for an organization with sufficient resources. An exhaustive keysearch is, at least in principle, possible.

– The design criteria of the tables used in the f-function are not known. Statistical tests however show that these tables are not completely random. Maybe there is a hidden trapdoor in their structure.

During the first twenty years after the publication of the DES-algorithm no effective way of breaking it was published. However, in 1997, for the first time, a DES challenge has been broken by a more or less brute-force attack. [1]

AES (Advanced Encryption Standard)

After breaking DES in 1997, same year NIST (National Institute of Standards and Technology) announced Advanced Encryption Standard (AES) competition to replace the Data Encryption Standard (DES). The final requirements specified a block cipher with 128-bit block size and support for 128, 192 or 256-bit key sizes. Evaluation criteria included security, performance on a range of platforms from 8-bit CPUs (e.g. in

smart cards) up, and ease of implementation in both software and hardware. In 2000 the Rijndael was announced to be a winner. It was designed by two Belgians, Joan Daemen and Vincent Rijmen. It is an iterated block cipher, but not a Feistel cipher; the overall structure is an substitution-permutation network. Nonlinearity is obtained by mixing operations from different algebraic groups.

Rijndael parameters

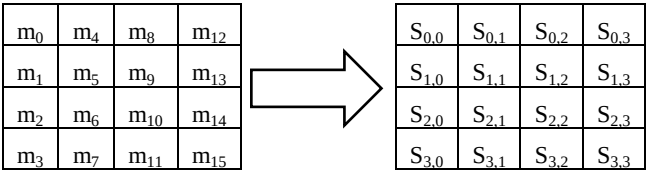
- Length of plaintext and ciphertext block is 128, 192, 256 bits.
- Key length is 128, 192, 256 bits.
- Number of rounds (N_b) is 10,12,14 (see table 2.2).

Table 2.2 Dependence of key and plaintext blocks lengths [2]

Key \ Text	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

Let consider the easiest case AES-128 where length of plaintext and ciphertext block equals to 128 bits. Number of rounds equal to 10.

AES operates on a 4 × 4 column-major order matrix of bytes, termed the state. 16 bytes (128 bits) of plaintext are used to initialize the state table by being written column by column.



Encryption in AES is performed iteratively using following operations:

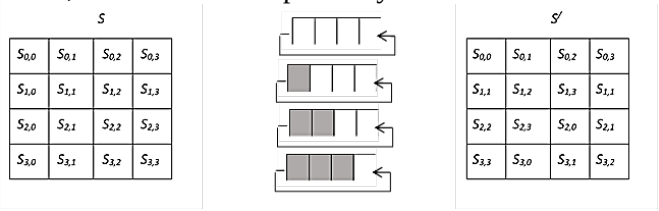
1. SubBytes – byte substitution table – nonlinear transformation (s-block)

In the *SubBytes step*, each byte $s_{i,j}$ is substituted with $s^*_{i,j}$ by looking up a fixed table (Rijndael S-box) given in design. This operation provides the non-linearity in the cipher. The S-box used is

generated by combining the inverse function (the multiplicative inverse over GF(2^8)) with an invertible affine transformation. The result is in a matrix of four rows and four columns.

2. ShiftRows – table transformation – cyclic shift each row of state by a fixed amount

The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in row *n* by *(n-1)* bytes. (For AES, the first row is left unchanged. Each byte of the second, third and fourth row is shifted by offsets of one, two and three respectively.



3. MixColumns – table transformation – data mixture in each column of state

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column.

4. AddKey – Cryptographic transformation – adding by modular 2 the round key and current state.

At the beginning we use the first key to randomise the state by operation XOR, then 9 same iterations performed. The last round (no MixColumns) is a bit different. The general algorithm is given on Fig.2.11.

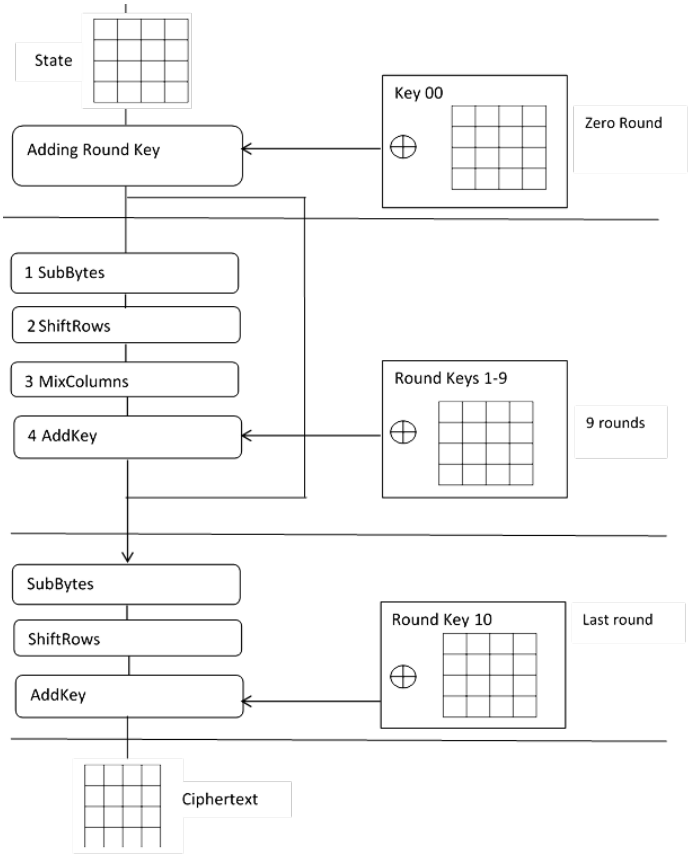


Fig.2.11 The general algorithm of AES

Key expansion procedure

Initially key in written into table like plaintext

w_0	w_1	w_2	w_3
$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

First round key is the real AES key. All other round keys are generated recursively by the rules:

– If number i is not multiple of N_b, then

$$w_i = w_{i-1} \oplus w_{i-N_b}$$

– If number *i* is multiple of N_b ($i : N_b$), then

$$w_i = \text{SubBytes}(\text{ShiftCol}(w_{i-1}) \oplus R_i) \oplus w_{i-N_b}$$

The procedure *Subbytes* means usage of cipher S-block to each byte of key, operation *ShiftCol* is a cyclic shift up by one position. R_i is a round constant.

Unlike the Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related. All transformations which used for decryption are inverse to correspondent encryption transformation. Each round consists of the four processes conducted in the reverse order with relevant round keys.

– Add round key.
– Inv Mix columns – mixture of the byte in column using the inverse matrix.
– Inv Shift rows – cyclic shift bytes to the right.
– Inv Byte substitution – substitution operation that used inverse table SubBytes^{-1}.

2.6 Foundations of Public Key Encryption

Symmetric cryptography has few following serious drawbacks:

– the key management problem (too many keys);
– the key exchange problem (the necessity of usage the secure channel prior communication);
– the trust problem (authenticity problem).

Public key cryptography was originally invented to solve given problems. The concept of public key cryptography was first presented in the paper of Diffie and Hellman entitled “New Directions in Cryptography” in 1976 [4]. The idea behind public-key cryptosystem is to replace two identical keys for encryption and decryption with two types of keys. Public key is used for encryption and could be published in some directory to be seen by everyone; secret (private) key is used in decryption scheme by its personal owner. Two keys are linked in a mathematical way, such that public key tells nothing about private key. More formally, it could be described in terms of one-way functions which are central to public-key cryptography.

f(x) is called to be one-way function if for given *x* it is easy to compute *f(x)*, but given *f(x)* it is hard to compute *x*. Here, "easy" and "hard" are to be treated in the sense of computational complexity theory, specifically the theory of polynomial time problems.

Diffie and Hellman invented key exchange protocol which allowed to establish one common secret key between two parties by exchanging some nonclassified information. This protocol uses same named one-way function $y=\alpha^x \ mod \ p$, where p is a large prime number, α is a primitive root modulo p.

Alice and Bob have to declare α and p as public parameters. Then Alice chooses her secret x_a and Bob selects his secret number x_b. They transmit over public channel to each other computed values: $y_a=\alpha^{x_a} \bmod p$ and $y_b=\alpha^{x_b} \bmod p$. The common secret key can be computed by both sides as $y_{ab} = y_a^b \ mod \ p$ or $y_{ab} = y_b^a \ mod \ p$. This scheme is illustrated on fig.2.12.

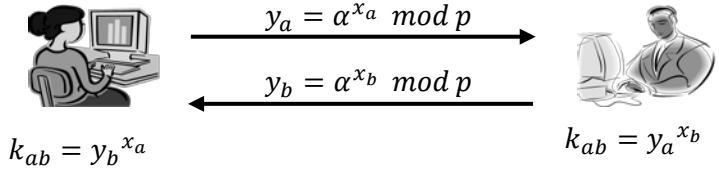


Fig. 2.12 Diffie-Hellman protocol

Despite of the fact that one-way functions have a wide application (cryptography, personal identification, authentication, e-commerce, e-banking and so on), a message encrypted with the one-way function is not useful; no one could decrypt it.

A trapdoor one-way function is a special type of one-way function, one with a secret trapdoor. It is easy to compute *f(x)* given *x*, and hard to compute *x* given *f(x)*. However, there is some secret information, *k*, such that given *f(x)* and *k* it is easy to compute *x*.

Although Diffie and Hellman invented the concept of trapdoor one-way function, it was not until a year or so later that the first (and most successful) system, namely RSA, was invented. RSA is named after the three inventors—Ron Rivest, Adi Shamir, and Leonard

Adleman. Security of RSA is based on the difficulty of factoring large numbers.

RSA algorithm can be briefly described by following steps [3]:

1. To generate two large prime numbers *p* and *q*.
2. To compute $n=p \cdot q$.
3. To select randomly integer e such that greatest common divisor of *e* and *(p-1)(q-1)* equals to 1 (*e* and *(p-1)(q-1)* are relatively prime).
4. Using extended Euclidean algorithm to compute the decryption key d such that $e \cdot d \equiv 1 \bmod (p-1)(q-1)$.
   In other words $d=e^{-1} \bmod (p-1)(q-1)$.
5. The numbers *(n,e)* are public key
   The numbers *(p,q,d)* are used as secret key.
   Numbers p and q are not need anymore but should be kept in secret.
   The simple formula $C=M^e \bmod n, M<n$ can be used for encryption now. To decrypt the message we can use formula $M=C^d \bmod n$.

It is obvious that both symmetric and asymmetric methods have a great impact on development of computer networks, internet and other communication means.

2.7 Resilient cryptography

In traditional cryptography, primitives are treated as mathematical objects with a predefined (well-restricted) interface between the primitive and the user/adversary. Based on this view, cryptographers have constructed a plenty of cryptographic primitives (CPA/CCA secure encryption schemes, identification schemes, unforgeable signatures, etc.) from various computational hardness assumption.

Cryptography, however, should be developed for actual deployment in real-world applications and not solely for theoretical purposes. In this new setting, the actual interaction between the primitive and the adversary depends not only on the mathematical description of the primitive, but also on its implementation and the specifics of the physical device on which the primitive is implemented. The information about the primitive leaked to the adversary goes well

beyond that predicted by the designer and, accumulatively, can allow the adversary break an, otherwise secure, primitive. Let consider some attacks based on physical attributes of a computing device which can reveal some information about internal secret key.

Types of Side Channel Attacks

Timing Attacks are one of the first type of such attacks which uses the running time of the execution of a protocol in order to obtain confidential information of user. The adversary knows a set of messages as well as the running time the cryptographic device needs to process them. He can use these running times and potentially (partial) knowledge of the implementation in order to derive information about the secret key. This attack was presented by Paul Kocher in [5], where he describe the results of experiment of timing attack on modular exponentiation and multiplication in RSA on the example of smart cards. The results of the experiment for implementing RSA on a smart card were reported by Schindler and others [6]

OpenSSL is a well-known open source cryptographic library that is often used on Apache web servers to provide SSL functionality. Brumley and Boneh [7] demonstrated that time attacks can reveal RSA private keys from a Web server based on OpenSSL over a local area network.

Power Analysis Attacks: In this kind of attacks, the adversary gets side information by measuring the power consumption of a cryptographic device. Power analysis attack is especially effective in attacks on smart cards or other special embedded systems storing a secret key. In cryptographic implementations where the execution path depends on the processed data, the power trace can reveal the sequence of instructions executed and hence leak information about the secret key. Various examples of power analysis attacks were demonstrated firstly by Kocher in [8]. Power analysis attacks were demonstrated as very powerful attacks for the simplest implementations of a symmetric and asymmetric ciphers in more than 200 papers.

Fault Injection Attacks: These attacks fall into the broader class of tampering attacks. The adversary forces the device to perform erroneous operations (i.e. by flipping some bits in a register). Generally speaking the fault injection attack requires two main steps: the injection of a fault and usage of steps with erroneous operations. If the

implementation is not robust against fault injection, then an erroneous operation might leak information for the secret key. The most common methods of influence are described in [9]. For instance, failures in a smart card can be caused by an impact of environment and placing it in an emergency condition.

Memory Attacks: This type of attack was recently introduced by Halderman in [10]. It is based on the fact that DRAM cells retain their state for long intervals even if the computer is unplugged. Hence an attacker with physical access to the machine can read the content of a fraction of the cells and deduce useful information about the secret key. Halderman et. al. studied the effect of these attacks against DES,AES and RSA. In October 2005, Dag Arne Osvik, Adi Shamir and Eran Tromer presented a paper demonstrating several cache-timing attacks against AES [11].

There are some more less known side channel attacks but leakage-resilient cryptosystems should remain secure even if the attacker learns some arbitrary partial information about their internal secret key through the physical or other leakages. There are some countermeasures can be applied in order to make cryptosystem resilient to side-channel attacks: adding noise, aligning the running time, balancing energy consumption, masking or blinding computing.

Questions to self-checking

1. Classify the cryptographic methods.
2. List types of cryptographic attacks.
3. What a difference between perfect and computational secrecy?
4. Give an example of cipher with perfect secrecy.
5. How to describe LFSR?
6. A5/1 – stream cipher based on LFSR.
7. Data encryption standard (DES).
8. Main operations in AES.
9. Advantages and disadvantages of symmetric cryptography.
10. What Diffie-Hellman protocol can be used for?
11. RSA as one of the best algorithms of asymmetric cryptosystem.
12. List types of side-channel attacks.
13. Describe the notation of resilient cryptography.

References

1. Henk C.A. van Tilborg, Sushil Jajodia (editors) Encyclopedia of Cryptography and Security. Second edition– Springer, Netherlands 2005. – 684 p.
2. Nigel Smart. Cryptography Made Simple. – Springer, 2016. – 481 p.
3. Gorbenko I.D., Gorbenko Yu.I. Applied Cryptology . Theory. Practice. Application, Kharkiv, Ukraine, Fort Publisher, 2012. – 880 p.
4. Diffie W.; Hellman M. New Directions in Cryptography // IEEE Transactions on Information Theory, Vol. IT-22, No. 6, November 1976. – pp.644-654
5. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellmann, RSA, DSS, and other systems // Advances in Cryptology – CRYPTO '96 :– Springer, 1996. – Vol. 1109. – pp.104–113.
6. W. Schindler. A timing attack against RSA with the Chinese Remainder Theorem // Proc. of Cryptographic Hardware and Embedded Systems (CHES 2000), Springer, 2000, LNCS 1965. – pp.109-124
7. D. Brumley, D. Boneh. Remote Timing Attacks are Practical // Proceedings of the 12th Usenix Security Symposium (August 4–8, 2003), 2003. – pp. 1-13.
8. P. Kocher, J. Jaffe, B. Jun. Differential power analysis. CRYPTO’99,LNCS 1666, 1999. – pp.388-397.
9. Jean-Jacques Quisquater, Francois Koeune (2010-10). Side Channel Attacks. State-of-the-art // CRYPTRECK 2002. – pp. 12-13
10. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In Paul C. van Oorschot, editor,USENIX Security Symposium,2008. – pp. 45–60
11. Dag Arne Osvik; Adi Shamir; Eran Tromer Cache Attacks and Countermeasures: the Case of AES // Proceeding CT-RSA'06

Proceedings of the 2006 The Cryptographers' Track at the RSA conference on Topics in Cryptology. – pp.1-20

3 POST-QUANTUM CRYPTOGRAPHY

3.1 Quantum computers and their impact on cryptography

Quantum computers, obeying the laws of quantum mechanics, can calculate things in ways that are unimaginable from the perspective of people's regular day-to-day experiences. In classical computing, information is stored in fundamental units called bits, where a bit can hold a binary digit with the value of 0 or 1. In quantum computing, the fundamental unit can hold both a 0 and a 1 value at the same time; this is known as a superposition of two states. These quantum bits are known as qubits and measuring the state of a qubit causes it to select or "collapse into", being a 0 or a 1. Interestingly, if you prepare a string of qubits of the same length in the same way, the resulting bit string will not always be the same. This gives quantum computers the advantage in form of the ability to perform very rapid parallel computations. Using these properties, a quantum computer is able to solve certain problems like searching and factoring much faster than a classical computer [1].

The most widespread asymmetric cryptosystems, like RSA and ElGamal ciphers, digital signature schemes (for example, DSA, ECDSA, etc.) and key-exchange protocols, which include Diffie-Hellman and ECDH schemes, are built on top of the mathematical complexities of integer factorization and discrete logarithms, which are considered to be NP-problems for classical computers. The algorithms for quantum computer, proposed by mathematician Peter Shor, are able to solve these problems in polynomial time. Thus, a large-scale quantum computer would be able to break all the mentioned above cryptosystems [1].

Shor's algorithm for integer factorization includes the five steps, among which only the second one requires performing quantum computations. The other steps are intended for execution on a classical computer. According to this algorithm, factorization of the positive composite integer N includes the following operations [2]:

1. Random choice of integer m which is strictly between 1 and N. Computation of the greatest common divisor gcd(m, N) of m and N using the polynomial time Euclidean algorithm. If $\gcd(m, N) \neq 1$ then a non-trivial factor of N has been found and algorithm returns gcd(m, N).

2. Determination of the period p of the function $f(x) = m^x \bmod N$ by means of quantum computer. Shor's algorithm solves this problem in

polynomial time.

3. If p is odd, the go to the first step. The probability of this is 0.5^k where k is the number of distinct prime factors of N.

4. Since p is even $m^p - 1 = (m^{0.5p} - 1)(m^{0.5p} + 1) = 0 \pmod N$. If $m^{0.5p} \bmod N = -1$ then go to the first step. The probability of going backward is less than 0.5^{k-1}, where k denotes the number of distinct prime factors of N.

5. Return of $\gcd(m^{0.5p} - 1, N)$ which is computed by the Euclidean algorithm. Since $m^{0.5p} \bmod N = -1$, it can be shown that the returned value is a non-trivial factor of N.

Shor's algorithm for discrete logarithm is able to solve the problem of finding the least integer x such that $g^x \bmod p = n$, where p is a large prime integer, g is a generator of the multiplicative group modulo p and n is positive integer which is less the p. In this case the bivariate function $f(a, b) = g^a \cdot n^{-b} \bmod p$ is being considered. This function has two-dimensional period, which equals (x, 1), because $f(a + x, b + 1) = g^{a+x} \cdot n^{-b-1} \bmod p = g^a \cdot n^{-b} \cdot g^x \cdot n^{-1} \bmod p = f(a, b)$. The considered algorithm uses a quantum computer to find this period in polynomial time and thus determine the value of x [3].

The experts in field of cryptography are starting to catch sight of quantum computing being a matter of near future. A large number of major cryptography specialists in IT industry are of opinion that a fully fledged quantum computer can be constructed in less than 10 years [4]. Moreover, the security of information, which was encrypted by quantum-unsafe cryptosystems, can be achieved only if the time period, during which these data must be kept in secret, is ended before creation of large-scale quantum computer [1]. These circumstances increase the actuality of post-quantum cryptography.

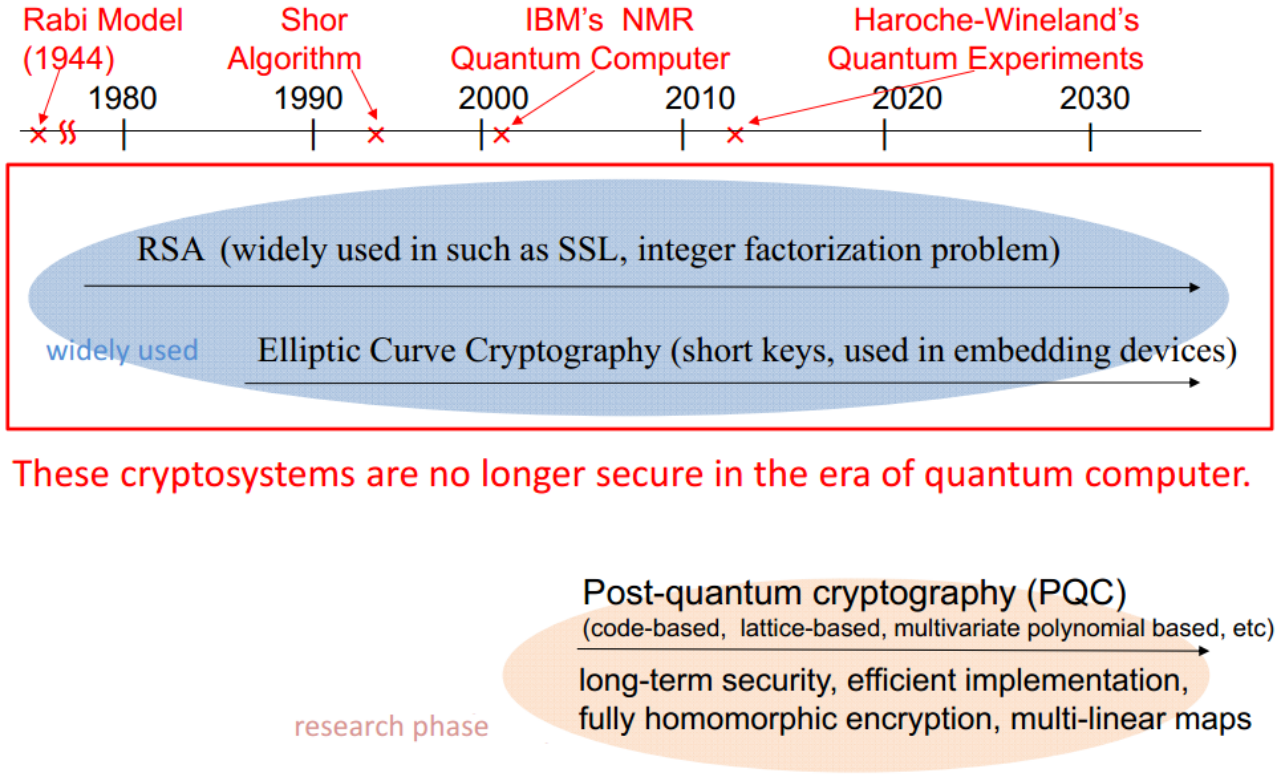


Figure 3.1 History of public key cryptography [5].

3.2 Post-quantum cryptography concept

Post-quantum cryptography is a variety of cryptographic algorithms resistant to attacks using quantum computations. The development of such cryptography requires the investigation of computational problems which cannot be solved in polynomial time by both classical and quantum computers. The main classes of such problems stem from the fields of lattice theory, coding theory and the study of multivariate quadratic polynomials. Each of these classes offers new possible frameworks within which to build public key cryptography. The quantum-safe ciphers that are built on these methods do admittedly present some challenges. Typically, they suffer from large key sizes when compared to popular, current public key algorithms that are not quantum-safe. However, in terms of performance, some quantum-safe algorithms are competitive with – or even faster than – widely used public key algorithms such as RSA or elliptic curve cryptosystems [1].

Some forms of symmetric-key cryptography are guaranteed to be quantum-safe. These primitives make no computational assumptions and are thus information-theoretically secure. An example of this is Vernam's One Time Pad, which has been proven to have perfect unconditional security against arbitrarily powerful eavesdroppers. Wegman-Carter Authentication is also known to be resistant against quantum attacks [1].

There are also other types of symmetric key cryptography that are believed (but not proven) to be quantum-safe. For example, generic quantum search only provides a quadratic speedup over classical search, indicating that quantum computers could not perform a brute force search to find symmetric keys much faster than could classical computers. Thus, unless the symmetric key algorithm happens to have a particular structure that can be exploited by a quantum computer, the bit security of a symmetric cipher can be retained in the presence of a quantum adversary by simply doubling the key length. Since quantum search does not provide exponential speedups, symmetric key encryption like AES is believed to be quantum-safe. The similar can be stated about good hash functions [1].

Post-quantum asymmetric cryptosystems, which do not use the quantum properties, include the following classes:

1. Code-based cryptosystems where the security depends on the

difficulty of solving a decoding problem in a linear code [6]. While there have been some proposals for code-based signatures, code-based cryptography has seen more success with encryption schemes [7]. The classic example is McEliece's public-key encryption system based on the hidden binary Goppa codes [8].

2. Multivariate cryptosystems where the security depends on the difficulty of solving a system of multivariate polynomial equations over finite fields [6]. While there have been some proposals for multivariate encryption schemes, multivariate cryptography has historically been more successful as an approach to signatures [7]. One of many interesting examples is Patarin's "HFEv−" public key signature system which generalizes a proposal by Matsumoto and Imai [8].

3. Lattice-based cryptosystems where the security depends on the difficulty of solving a short or close vector problem in a lattice [6]. Most lattice-based key establishment algorithms are relatively simple, efficient, and highly parallelizable [7]. The example that has perhaps attracted the most interest, not the first example historically, is the Hoffstein–Pipher–Silverman "NTRU" asymmetric cryptosystem [8].

4. Hash-based signatures where the security depends on the difficulty of finding collisions or preimages in cryptographic hash functions [6]. Many of the more efficient hash-based signature schemes have the drawback that the signer must keep a record of the exact number of previously signed messages, and any error in this record will result in insecurity. Another of their drawbacks is that they can produce only a limited number of signatures. The number of signatures can be increased, even to the point of being effectively unlimited, but this also increases the signature size [7]. The classic example is Merkle's hash-tree public-key signature system (1979), building upon a one-message-signature idea of Lamport and Diffie [8].

5. Isogeny-based cryptosystems where the security depends on the difficulty of finding an unknown isogeny between a pair of supersingular elliptic curves. They have good properties such as small key sizes and forward security. These cryptographic schemes are a new research field with relatively few active research groups or publications and deserve more academic scrutiny to establish a consensus on their security properties. The classic example is Jao-De Feo key agreement protocol [6].

3.3 Code-based cryptography

The main advantage of this class of cryptosystems besides the quantum resistance is high performance. Its biggest drawback is a large key size which is the reason of less prevalence of these cryptosystems in comparison with RSA and ElGamal schemes [1].

Classical code-based ciphers, like the McEliece and Niederreiter schemes, can be built on top of different linear error-correction codes, but not all of them are suitable to achieve proper security of the obtained cryptosystems. As an example the Niederreiter scheme based on generalized Reed-Solomon codes can be given, which was broken by Sidelnikov and Shestakov with a structural attack. Nevertheless, these cryptosystems based on the hidden binary Goppa codes are secure against attacks by both classical and quantum cryptanalysis [9].

The binary Goppa codes play an important role in ensuring the cryptographic strength of the McEliece and Niederreiter ciphers. Breaking these cryptosystems requires decoding of arbitrary public linear code which was obtained by random transformation of some quickly decodable hidden one. This is an NP-complete problem [9].

There is no known efficient algorithm which allows to distinguish between the binary Goppa code and a binary random one [10]. The number of inequivalent binary Goppa codes grows exponentially with the increase of their length and dimension [11]. These circumstances make impossible to recover in polynomial time a hidden code and reduce the subsequent stage of break of the McEliece and Niederreiter cryptosystems to an effective decoding of the restored code [9].

3.3.1 The McEliece cryptosystem

This cryptosystem has been developed by Robert McEliece in 1978 and become the first probabilistic encryption scheme [12]. This cipher uses the mathematical apparatus of matrices and vectors over a field GF(2) and binary linear codes.

The key pair generation includes the following actions [9]:

1. Random choice of a binary linear (n, k)-code C with $k \times n$ generator matrix G. This code must be able to correct no less than t errors and have an efficient decoding algorithm.

2. Random generation of $n \times n$ permutation matrix P and nonsingular $k \times k$ matrix S.

3. Computation of k × n matrix $E = S \cdot G \cdot P$.

4. Creation of a pair of public and private keys in the form of tuples (E, t) and (S^{-1}, G, P^{-1}), respectively.

The encryption consists of the following steps [9]:

1. Representation of a message as k-dimensional vector m.

2. Calculation of a vector $v = m \cdot E$.

3. Choice of a random n-dimensional vector z of weight t.

4. Forming of a ciphertext by the formula $c = v + z$.

The decryption requires the following operations [9]:

1. Computation of n-dimensional vector $u = c \cdot P^{-1}$.

2. Obtainment of vector d as a result of decoding of u with C.

3. Forming of a decrypted message vector $m' = d \cdot S^{-1}$.

The correctness of this cipher can be proved as follows:

1. A vector u equals $c \cdot P^{-1} = (m \cdot E + z) \cdot P^{-1} = m \cdot S \cdot G + z \cdot P^{-1}$. Since G is a generator matrix of the linear code C, a vector $m \cdot S \cdot G$ is a word of C. A vector $z \cdot P^{-1}$ is of weight t, because P is a permutation matrix. Thus, a vector u is a result of distortion of t symbols in a word of C, which was obtained by encoding of a message $m \cdot S$.

2. A vector d, which is a result of decoding of u with C, has a value $m \cdot S$ as this code allows to correct no less than t errors.

3. A decrypted message vector m′ equals $d \cdot S^{-1} = m$.

3.3.2 The Niederreiter cryptosystem

This cryptosystem has been proposed by Harald Niederreiter in 1986. Unlike the McEliece cryptographic scheme, it is deterministic, but has a higher speed of encryption and can be used to create a digital signature [13]. However, the generation time for such signatures is more than for other quantum-safe ones [1]. This cipher uses the same mathematical apparatus as the McEliece cryptosystem.

The key pair generation consists of the following steps [9]:

1. Random choice of a binary linear (n, k)-code C with $(n - k) \times n$ parity check matrix H. An efficient decoding algorithm for C must be known. The error correction capability of C should be no less than t.

2. Random generation of $n \times n$ permutation matrix P and nonsingular $(n - k) \times (n - k)$ matrix S.

3. Calculation of $(n - k) \times n$ matrix $E = S \cdot H \cdot P$.

4. Creation of a pair of public and private keys in the form of tuples (E, t) and (S^{-1}, H, P^{-1}), respectively.

The encryption requires the following operations [3]:

1. Representation of an initial message as n-dimensional vector m of weight t.

2. Forming of a ciphertext by the formula $c = E \cdot m^T$.

The decryption includes the following actions [3]:

1. Computation of (n – k)-dimensional vector $u = S^{-1} \cdot c$.

2. Determination of a variable d as a transposed error vector which corresponds to a syndrome u in the error correction procedure of C.

3. Forming of the decrypted message vector $m' = (P^{-1} \cdot d)^T$.

The correctness of this cipher can be justified as follows:

1. A vector u equals $S^{-1} \cdot c = S^{-1} \cdot E \cdot m^T = H \cdot P \cdot m^T$. The weight of a column vector $P \cdot m^T$ equals t, because P is a permutation matrix. Since H is a parity check matrix of the linear code C, u is a syndrome of an error vector $(P \cdot m^T)^T$, which has a weight t.

2. A vector d, which is the result of calculation of the transposed error vector which corresponds to a syndrome u in the error correction procedure of the code C, is equal to $P \cdot m^T$, because C is able to correct no less than t errors.

3. A decrypted message vector m′ has a value $(P^{-1} \cdot d)^T = m$.

3.4 Multivariate cryptography

The primary application field of this class of cryptosystems is digital signature schemes [7]. The main advantages of the multivariate signature schemes are very short signatures and high performance [14]. Their biggest shortcoming is a large key size [8].

To break these cryptosystems one must find a solution of arbitrary public system of multivariate quadratic equations over a finite field. In general, this problem is NP-hard. In multivariate cryptographic schemes the public equation system is obtained by two hidden random linear transformations of some private vector-valued quadratic map, which is easy to invert. Different families within this class of cryptosystems correspond to several approaches to construction of the aforementioned private map [14].

The first multivariate cryptosystem was C* scheme, which has been designed by Matsumoto and Imai in 1985. An approach to break it has been proposed by Patarin in 1995. Other well-known cryptosystems of this class include HFE, UOV, Rainbow and IFS. Some multivariate schemes were broken, but other remain secure both in classical and

post-quantum senses [14].

The key pair generation includes the following actions [14]:

1. Random choice of a central map Q which must be an easily invertible quadratic map between vectors over finite field F. Each n-dimensional vector over F must be mapped by Q to m-dimensional vector over the same finite field. The central map can be represented as m-tuple of n-variate polynomials $q_i(x_1,...,x_n)$ where only quadratic terms are present. The mapping of $(v_1,...,v_n)$ to $(u_1,...,u_m)$ can be performed according to the formula $u_i = q_i(v_1,...,v_n)$.

2. Random choice of initial and final maps S and T, which must be affine maps between vectors over F. The dimensions of mapped vectors are n for S and m for T. The initial map can be represented as n-tuple of linear n-variate polynomials $s_i(x_1,...,x_n)$. The mapping of $(v_1,...,v_n)$ to $(u_1,...,u_n)$, which corresponds to S, can be performed by the formula $u_i = s_i(v_1,...,v_n)$. The similar holds true for the final map. The mapping formula for T is $u_i = t_i(v_1,...,v_m)$ where $t_i(x_1,...,x_n)$ are polynomials of m-tuple representation of the final map.

3. Representation of a map composition P, which equals $T \circ Q \circ S$, as m-tuple of n-variate quadratic polynomials $p_i(x_1,...,x_n)$. The mapping of $(v_1,...,v_n)$ to $(u_1,...,u_m)$, which corresponds to P, is described by the formula $u_i = p_i(v_1,...,v_n)$. The coefficients of the polynomials in the aforementioned m-tuple can be obtained from the identity $p_i(x_1,...,x_n) = t_i(q_1(s_1(x_1,...,x_n), ..., s_n(x_1,...,x_n)), ..., q_m(s_1(x_1,...,x_n), ..., s_n(x_1,...,x_n)))$.

4. The public key is created in the form of mentioned above polynomial representation of P. The private key must is represented by the tuple (T^{-1}, Q^{-1}, S^{-1}), where the elements correspond to inverse maps.

The message encryption or verification of digital signature can be performed as follows [14]:

1. An initial message or a digital signature is represented as n-dimensional vector d over F.

2. A vector r is obtained as the result of applying of P to d. A ciphertext is represented by r if an encryption is performed. In case of verification a signature is considered invalid if r does not represent the hash of signed message.

The decryption or signing of message consists of the following operations [14]:

1. A ciphertext or a hash of signed message is represented as m-dimensional vector r over F.

2. A vector d is computed as the result of applying of map composition $S^{-1} \circ Q^{-1} \circ T^{-1}$ to r. A decrypted message or digital signature is represented by d.

The correctness of these cryptosystems can be proved as follows. The decryption of ciphertext with a correct private key yields the result of applying of a map composition $S^{-1} \circ Q^{-1} \circ T^{-1} \circ P$ to an initial message vector. Since $S^{-1} \circ Q^{-1} \circ T^{-1} \circ P = S^{-1} \circ Q^{-1} \circ T^{-1} \circ T \circ Q \circ S$, the aforementioned map composition is equal to an identity map. Thus, the decryption produces an initial message vector if a corresponding private key is used. The similar approach can be applied to prove that a signature, which was created with a correct private key, is considered valid by the verification procedure.

3.5 Comparison of post-quantum and classical cryptosystems

The following tables, which were given in the referred ETSI White Paper, compare the practical factors between public key cryptography schemes that are popular, but vulnerable to quantum attack, and quantum-safe public key schemes. The important factors being compared include key generation time, signing time, verification time, encryption time, and decryption time. The data represented in the tables is not benchmark data, but instead are values that are relative to an RSA signing operation where 1 unit of time is equivalent to producing an RSA signature using a 3072 bit private key [1].

The time values are extrapolated from ECRYPT Benchmarking of Cryptographic Systems and the papers specifying the schemes, which were selected for comparison in the aforementioned ETSI document. In addition to comparing the time taken to perform cryptographic operations, the key sizes of the public key and private key, and the size of the resulting the cipher text are shown. These comparisons all assume an equivalent effective symmetric key strength of 128 bits and are represented by the value k (i.e. k = a key that is as strong as a 128 bits of symmetric key). The time scaling and key scaling columns describe the rate at which operation time increases and the size of keys increase in order to increase the security level [1].

Hash tree based signatures are unique in that their keys can only be used to produce a limited number of signatures. The maximum number of signatures for a hash tree scheme needs to be chosen at the time of key generation. For the purpose of comparisons below, hash tree

scheme keys are set at a fixed 2^{20} signatures [1].

The following table comparisons are not exact and are intended for illustration only. Currently, the actual implementation benchmarks for these quantum-safe schemes are not generally available. The data on performance provided in Table 1 and Table 2 is based on estimations to obtain approximate scaling about the performance and is not the result of tests conducted in the same controlled environment. Thus, the performance data should not be considered as a precise comparison [1].

Based on Table 1, the key pair generation of the selected quantum-safe schemes are far better than RSA, but not as good as DH and ECDH. Thus, using a one-time key pair to achieve perfect forward secrecy is possible during a key establishment scheme, however, it will be slower than an ephemeral Diffie-Hellman key agreement [1].

For the selected signature schemes, XMSS has the asymmetry property of RSA, i.e. verifying is faster than signing. Likewise, for the selected encryption schemes, the McEliece variants also share this property of RSA, i.e. encryption is faster than decryption [1].

The selected quantum-safe schemes generally have performance comparable to or better than pre-quantum schemes of the same security level. However, key, message, and signature sizes are generally larger. In the cases of McEliece and Rainbow, key sizes are a lot larger. Also, quantum-safe schemes have not been studied as thoroughly as the listed pre-quantum schemes [1].

Table 3.1 Comparison on encryption schemes [1].

Algorithm	KeyGen (time compared to RSA decrypt)	Decryption (time compared to RSA decrypt)	Encryption (time compared to RSA decrypt)	PubKey (key size in bits to achieve 128 bits of security)	PrivateKey (key size in bits to achieve 128 bits of security	Cipher text (size of resulting cipher text)	Time Scaling	Key Scaling
NTRU	5	0.05	0.05	4939	1398	4939	k^2	k
McEliece	2	0.5	0.01	1537536	64861	2860	k^2	k^2
Quasi-Cyclic MDPC McEliece	5	0.5	0.1	9857	19714	19714	k^2	k
RSA	50	1	0.01	3072	24,576	3072	k^6	k^3
DH	0.2	0.2	0.2	3072	3238	3072	k^4	k^3
ECDH	0.05	0.05	0.05	256	256	512	k^2	k

Table 3.2 Comparison on digital signature schemes [1].

Algorithm	Num of sign	Key Gen (time compared to RSA sign)	Signing (time compared to RSA sign)	Verifying (time compared to RSA sign)	PubKey (size in bits to achieve 128 bits of security)	PrivateKey (size in bits to achieve 128 bits of security	Signature (size in bits of resulting digital signature)	Time Scaling	Key Scaling
XMSS signatures (hash based)	2^{20}	100000	2	0.2	7296	152	19608	k^2	k^2
BLISS (lattice-based)		0.005	0.02	0.01	7000	2000	5600	k^2	k
Rainbow signature (multivariate)		20	0.02	0.02	842400	561352	264	k^3	k^3
RSA		50	1	0.01	3072	24,576	3072	k^6	k^3
DSA		0.2	0.2	0.2	3072	3328	3072	k^4	k^3
ECDSA		0.05	0.05	0.05	512	768	512	k^2	k

Lattice-based schemes offer good security, relatively short keys, fast key generation for forward security and the flexibility to provide key agreement, key transport and key pre-distribution schemes [6].

Code-based schemes based on binary Goppa codes are well established and offer good security as basic key transport schemes. They are somewhat less flexible than lattices and may need supplementing to provide forward security or other features. The

various proposals to reduce key sizes for code-based schemes are very interesting but would benefit from more academic assessment [6].

Multivariate signature schemes uniquely offer very short signatures, which might be good for some use cases [6].

Hash-based signature schemes offer good security but their practical requirements for bookkeeping and limits on the number of signatures available mean that they are not suitable for general-purpose applications [6].

3.6 Application in security protocols

The majority of contemporary security protocols use asymmetric cryptosystems which are vulnerable to attacks performed on quantum computers. Thus, to achieve the quantum-safe implementation of these protocols, they should be rebuild on the basis of post-quantum cryptography. However, this task may be complicated because non-security issues such as adoption rates, backwards compatibility and performance characteristics must also be considered. Some protocols are too rigid and require fundamental messaging and data structure changes to safeguard them from quantum threats [1].

3.6.1 X.509 certificates

The X.509 standard specifies a common format for public key certificates, mechanisms for managing and revoking certificates, a set of valid attributes of certificates, and an algorithm for validating the certificates. X.509 is not a protocol but rather a suite of data formats and algorithms that collectively constitute a public key infrastructure. These certificates play a central role in the use of SSL/TLS on the Internet, as servers are authenticated to clients using X.509v3 certificates. Every web server supporting TLS must have a certificate, the vast majority of which are issued by one of the several hundred commercial certificate authorities that are recognized by major web browsers. X.509v3 certificates are also used in other contexts, including secure email (S/MIME), web services (XML Digital Signatures), and code signing [1].

Using quantum-safe algorithms and public keys in X.509 certificates does not require a change to the standard. The structure of these certificates is extensible and can be made to support quantum-safe

algorithms with relative ease. However, X.509 is a standard that is used in many other standards that would require an update to support the newly defined quantum-safe algorithm identifiers. For example, TLS would require new ciphersuites to be introduced [1].

3.6.2 Internet Key Exchange (IKE) version 2

Internet Key Exchange is a protocol used to establish keys and security associations for the purpose of setting up a secure virtual private network connection that protects network packets from being read or intercepted over a public Internet connection. This allows a remote computer on a public network to access resources and benefit from the security of a private closed network without compromising security [1].

The IKE protocol standard is rigid and does not permit VPN designers to choose beyond a small set of cryptographic algorithms. At present, none of the permitted algorithms are completely quantum-safe. Thus, any option to make IKE quantum-safe would require a change to the standard [1].

3.6.3 Transport Layer Security (TLS) version 1.2

The Transport Layer Security protocol, earlier versions of which were called the Secure Sockets Layer (SSL) protocol, establishes a protected tunnel between a client and server for transmission of application data. The handshake sub-protocol is used to perform server-to-client and optional client-to-server authentication, and to establish shared secret keys. Shared secrets are subsequently used in the record layer sub-protocol to encrypt and authenticate application data [1].

TLS is used to secure a variety of applications, including web traffic (the HTTP protocol), file transfer (FTP), and mail transport (SMTP). The design of TLS is largely independent of cryptographic algorithms, and allows for parties to negotiate ciphersuites (combinations of cryptographic algorithms to use). As of TLSv1.2, all cryptographic components (public key authentication, key exchange, hash functions, bulk encryption) can be negotiated, although generally all must be negotiated at once in a single ciphersuite rather than independently. Thus, any option to make TLS quantum-safe would

require a change to the standards by introducing new ciphersuites. Also, quantum-safe algorithms with large public keys or signatures may require additional changes to the standard to allow certificates with size exciding 16 MB be used [1].

3.6.4 Secure/Multipurpose Internet Mail Extension (S/MIME)

Secure/Multipurpose Internet Mail Extension is a standard for digital signatures and public-key encryption used to securely send email messages. It offers origin authentication, non-repudiation, data integrity, and confidentiality through use of digital signatures and message encryption. This standard is widely adopted throughout government and enterprise. S/MIME, and a similar scheme called OpenPGP, allow email to remain encrypted during the entire path from sender to recipient, meaning that at the email servers of both the sender and receiver, as well as the links between sender, sender's email server, recipient's email server, and receiver, the plaintext cannot be read or modified by an adversary. This contrasts with other protocols like SMTP over TLS and IMAP/POP3 over TLS which are used to secure the individual links between intermediate mail servers, but do not preserve end-to-end confidentiality and data integrity [1].

The S/MIME protocol has the potential to transition to quantum-safe cryptography. The S/MIME Capabilities attribute (which includes algorithms for signatures, content encryption, and key encryption) was designed to be flexible and extensible so that other capabilities added later would not break earlier clients. However, some very early versions of S/MIME may present backward-compatibility issues [1].

3.6.5 Secure Shell (SSH) version 2

Secure Shell version 2 is a cryptographic network protocol used to encrypt information sent over an insecure network such as the Internet. In essence, it relies on a client-server model to allow a user on one computer to remotely log-in, send commands, and transfer files on another computer, without compromise of data integrity or confidentiality. It has a wide range of uses, with some implementations of SSH (namely OpenSSH) enabling users to create fully encrypted VPNs. This allows users to treat a public network such as the Internet as

if it were a more secure, private network [1].

SSH is a secure remote-login protocol. It has pervasive and diverse applications, and can be used for a variety of purposes, including the construction of cost-effective secure Wide Local Area Networks, secure connectivity for cloud-based services, and essentially any other enterprise process that requires secure remote access to a server [1].

The SSH protocol was specified with a high level of cryptographic agility and allows servers and clients to negotiate the algorithms used for encryption, data integrity, authentication and key exchange. The addition of quantum-safe controls will not require significant changes to the base SSH protocol [1].

3.7 Conclusions

The rapid progress in building of quantum computers threatens the most widespread public key cryptographic schemes. The mathematical problems, which underlie their security, can be solved by quantum algorithms in polynomial time. For example, Shor's algorithms for integer factorization and discrete logarithm allow breaking such cryptosystems as RSA, DSA and Diffie-Hellman schemes. These circumstances increase the actuality of post-quantum cryptography, which is a variety of cryptographic algorithms resistant to attacks using quantum computations.

Symmetric key encryption and good hash functions are believed to be quantum-safe. Asymmetric post-quantum cryptosystems, which use only classical computing, include such classes as code-based, multivariate, lattice-based, hash-based and isogeny-based cryptographic schemes. Their main advantage is high performance, however, these cryptosystems typically suffer from large key sizes.

Multivariate cryptographic schemes are more successfully used to create digital signatures, which in this case are very short. Code-based cryptosystems are primarily applied for encryption. Hash-based cryptography provides an approach only for obtaining digital signatures. Lattice-based cryptosystems have fast generated keys of relatively small size and so are suitable to be used for key agreement. Isogeny-based cryptographic schemes, which are a new research field, possess a significant advantage in the form of very short public key.

The majority of security protocols use quantum-unsafe asymmetric cryptosystems and so should be rebuild on the basis of post-quantum cryptography. However, this task may be complicated because non-security issues such as adoption rates, backwards compatibility and performance characteristics must also be considered.

The directions of further research in field of post-quantum cryptography include the investigation of security properties of relatively new quantum-safe cryptographic schemes and search of approaches to obviate the drawbacks of these cryptosystems.

3.8 Questions

1. What are the basic differences between quantum and classical computers? How does qubit differ from bit?
2. What mathematical problems underlie the most widespread asymmetric cryptosystems? What quantum algorithms threaten such cryptography?
3. What is a post-quantum cryptography? What advantages and drawbacks do quantum-safe cryptosystems typically have in comparison with widespread public key cryptographic schemes?
4. What are the main classes of post-quantum asymmetric cryptosystems? What mathematical problems underlie their security?
5. What of these cryptography classes are primarily applied for encryption? Which of them are more successfully used for obtaining digital signatures?
6. Why do the binary Goppa codes have an important significance in code-based cryptography?
7. What are the basic differences between the McEliece and Niederreiter schemes? What are the main advantages and drawbacks of these cryptosystems?
8. What is the general approach for construction of equation system of a public key in multivariate cryptography?
9. What is the primary application field of multivariate cryptosystems? What are their main advantages and drawbacks?
10. What issues complicate the introduction of post-quantum cryptography into contemporary security protocols? Which of these protocols possess a low level of cryptographic agility?
11. What is the role of ciphersuites in the cryptographic agility of TLS? How do they impact on the perspective of using the quantum-safe

cryptosystems in TLS?

3.9 References

1. M. Campagna, et al., “ETSI White Paper No. 8. Quantum Safe Cryptography and Security”, European Telecommunications Standards Institute, 2015, 64 p.

2. S. J. Lomonaco, “Shor's Quantum Factoring Algorithm”, AMS Proceedings of Symposia in Applied Mathematics, 2002, vol. 58, 19 p.

3. M. Hayashi, S. Ishizaka, A. Kawachi, G. Kimura, T. Ogawa, “Introduction to Quantum Information Science”, Springer, 2015, 332 p.

4. T. L. Swaim, “Quantum Computing and Cryptography Today: Preparing for a Breakdown”, University of Maryland University College, 2012, 22 p.

5. T. Takagi, “Recent Developments of Post-Quantum Cryptography”, Workshop on Cyber Security between RHUL and Kyushu University, 2016, 22 p.

6. “ETSI GR QSC 001: Quantum-Safe Cryptography (QSC); Quantum-safe algorithmic framework”, European Telecommunications Standards Institute, 2016, 42 p.

7. L. Chen, et al., “NISTIR 8105 DRAFT. Report on Post-Quantum Cryptography”, National Institute of Standards and Technology, 2016, 15 p.

8. D. J. Bernstein, J. Buchmann, E. Dahmen, “Post-Quantum Cryptography”, Springer, 2009, 246 p.

9. E. Jochemsz, “Goppa Codes & the McEliece Cryptosystem”, Vrije Universiteit Amsterdam, 2002, 63 p.

10. T. P. Berger, P.-L. Cayrel, P. Gaborit, A. Otmani, “Reducing Key Length of the McEliece Cryptosystem”, Lecture Notes in Computer Science, 2009, vol. 5580, pp. 77-97.

11. P. Fitzpatrick, J. A. Ryan, “Enumeration of inequivalent irreducible Goppa codes”, Discrete Applied Mathematics, 2006, vol. 154, iss. 2, pp. 399-412.

12. S. Y. Yan, “Quantum Attacks on Public-Key Cryptosystems”, Springer, 2013, 214 p.

13. R. Lu, X. Lin, X. Liang, X. Shen, “An efficient and provably secure public key encryption scheme based on coding theory”, Security and Communication Networks, 2011, vol. 4, iss. 12, pp. 1440-1447.

14. L. Goubin, J. Patarin, B.-Y. Yang, “Multivariate

Cryptography", Encyclopedia of Cryptography and Security, 2011, pp. 824-828.

4 SOFTWARE SECURITY AND TECHNOLOGIES FOR RESILIENT COMPUTING

4.1 Understanding software security and resilient software

It is difficult enough today to find a business that would not work on software which objectives and priorities of use differ from organization to organization and client to client. But in any case you need reliable and stable applications. The Institute of Electrical and Electronics Engineers (IEEE) Reliability Society defines reliability as «The target at which software designers have always aimed: perfect operation all the time» [1-2]. However, this does not mean that every element of the system will work reliably all the time, which gives us an understanding of the difference between reliability and resiliency. Resilience in this context means that failures must be compartmentalized [3].

Why is resilient software so important and why not just say "fail-safe" or "failure-resistant"? In a literal sense, "failsafe" means that a system fails in a safe way. On the other hand, the resistive systems return to their original operating state. In most cases, the main way to restore functionality is to restart the entire system: the application, and then, if it did not work, the server, and also have to restart everything that depended on the failed component. On the contrary, a resilient system automatically shuts down the failover components and reintegrates them as soon as they stop working. So when we talk about software, resilience is the capacity to resist and recover from deliberate attacks, accidents, artificial or natural threats.

Software resilience is the ability of an application to react to troubles in one of its components and still provide the best possible service. Resiliency has become more important as business continue to fast implement software across multiple technology infrastructures [4-5]. In other words, if application is resilient it means that when faced with any of these changes or even with a combination of them, it must adapt itself while ensuring the observance of the safety criteria.

In contrast to the challenges of integration of the hardware with software, the abilities of the "hacker" to undermine desired system behavior also create a significant challenge to application resilience.

At the same time, most software development projects do not include any actions to ensure the security of software – threat modeling or safe developer training. Therefore, the two most common causes of violations of information security today are people and related programs. Most of the software is implemented without a real approach to building protection from the point of view of cyberattacks. This will continue until people are more aware of the risks that need to be taken into account at each stage of product development. It should be understood that there is not one 100% safe software, but all software can be designed, developed and deployed in a safe manner by implementing secure software development.

Secure software development includes certain actions and steps that need to be integrated into the life cycle of software development to create a safe and protected from malicious influences and software attacks. The chart Fig. 4.1 shows an example of a common structured way of creating software that fully meets the needs of the end user, which includes the implementation of appropriate security measures [6]. The efficiency of resilient software depends on its ability to forecast, absorb, fit to, and/or recover rapidly from a potentially disruptive event [7].

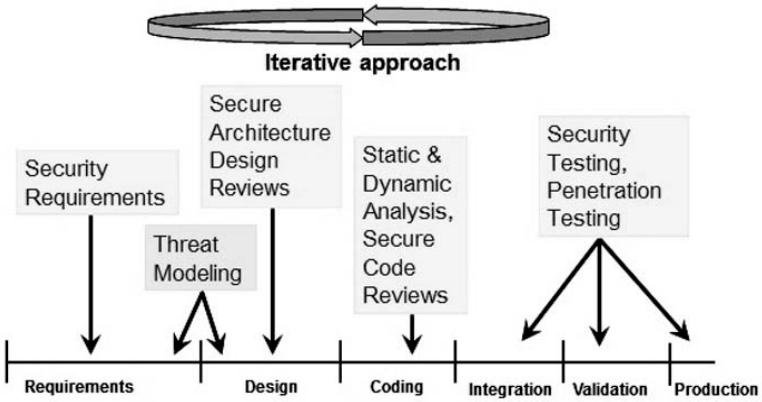


Fig. 4.1. Security in the SDLC process

Disregard the basic principles that lead to safe and sustainable software can be catastrophic for the enterprise and using of insecure

software can bring an organization to its knees. Because of fraudulent and computer crimes, companies around the world each year lose hundreds of millions of dollars and bear direct financial losses. For the eighth consecutive year, hacking/skimming/phishing attacks were the leading cause of data breach incidents, accounting for 55.5 percent of the overall number of breaches (Fig. 4.2) [8,9].

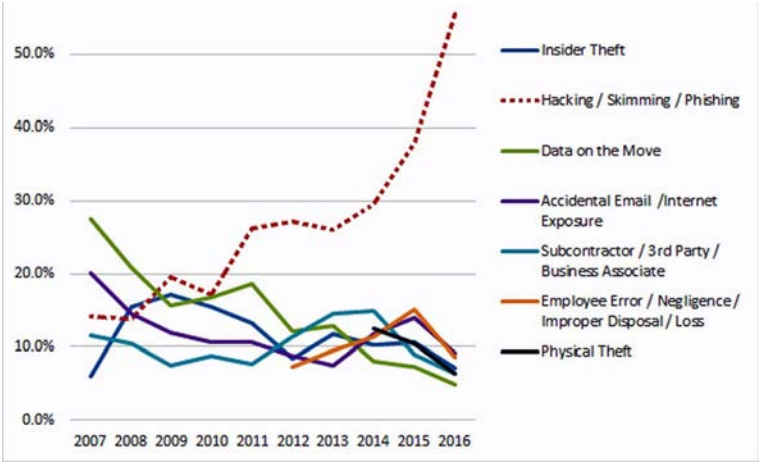


Fig. 4.2. Data breach incidents by type of occurrence

Just like software is everywhere, flaws in most of this application are everywhere too. Disadvantages in the software may threaten the security and safety of the systems on which they operate. These shortcomings are present not only on the traditional computers that we think about, but also on the most important devices we use, such as our cell phones and cars, pacemakers and hospital equipment, etc.

Taking into account the definition of the software development life cycle, as well as the survey of existing processes, process models and standards, it seems to determine the following four focus areas of SDLC for the development of secure software:

– security engineering activities;
– security assurance;
– security organizational and project management activities;
– security risk identification and management activities.

4.2 Designing applications for security and resilience

Secure by design, in software engineering, means that the software has been designed from the ground up to be secure. When designing applications, a number of practices are applied that will help improve the security of the application. First of all, it reduces the surface area of possible attacks (Attack Surface Reduction) and threat modeling. Despite the close relationship of these two concepts, the first mechanism implies an active reduction of the attacker's ability to exploit unknown security breaches. To reduce the area of possible attacks, you can use layered protection mechanisms and the principles of least privilege. Modeling the threats in turn allows us to guess which components of the system can be considered as vectors of attack. A convenient tool for threat modeling is the Microsoft Thread Modeling Tool based on the STRIDE classification.

At the design stage, you need to carefully study the security and privacy requirements associated with security issues and privacy risks. This process is called risk analysis. Risk analysis questions include the following [10]:

– threats and vulnerabilities existing in the project environment or arising from interactions with other systems;

– code created by external development groups in the source or object form. It is very important to carefully evaluate any code from sources external to your team. Otherwise, there may be security vulnerabilities that the project team does not know about;

– threat models should include all obsolete codes if the project is a new version of an existing program. Such code could be written before much was known about software security and therefore probably contains vulnerabilities;

– detailed confidentiality analysis to document the key aspect of the confidentiality of your project.

The software safety community has produced a number of robust tools for providing automated support in the traceability of complex requirements for safety-critical systems. Examples include Praxis High Integrity Systems' REVEAL, Telelogic's DOORS, ChiasTek's REQTIFY, Safeware Engineering's SpecTRM, etc.

Threat modeling is an iterative technique used to identify the threats. It starts by identifying the security objectives of the software as described in the security non-functional requirements (Fig. 4.3).

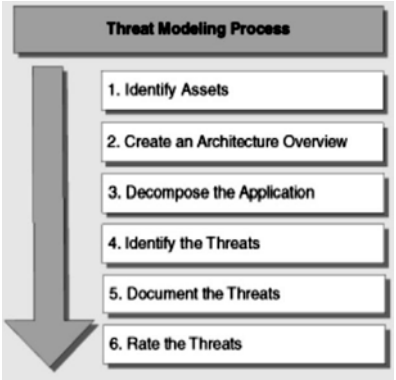


Fig. 4.3 Threat Modeling Process

Modeling is a well-known approach for detecting and studying-software requirements. It provides the opportunity to present the work and interaction of the proposed software within its intended environment. The closer the model reflects the intended environment, the more useful the modeling approach. Thus, the security of software development benefits from modeling, which explicitly includes security threats.

The main problems of modeling are: 1) doing it well; 2) do it carefully enough; 3) knowing what to do with the results, for example, how to convert analysis in the metric and / or in other cases a useful decision point. There are few threat, attack, and vulnerability modeling tools and techniques. Microsoft, for example, has emphasized this type of modeling in its secure software initiative.

The patterns were derived by generalizing existing best security design practices and by extending existing design patterns with security-specific functionality. Rather than focus on the implementation of specific security mechanisms, the secure design patterns detailed in this report are meant to eliminate the accidental insertion of vulnerabilities into code or to mitigate the consequences of vulnerabilities [11]. They are categorized according to their level of abstraction:

– architectural-level patterns;
– design-level patterns;
– implementation-level patterns.

In order to withstand the attack, the software must be clearly designed in accordance with the secure design principles. The following subset was suggested, which directly relates to software security [12]:

1) Minimize Attack Surface – reduce entry points that can be exploited by malicious users;

2) Least Privilege – just having enough access level to do the job;

3) Separation of Duties – different entities have different roles;

4) Defense in Depth – multiple layers of control make it harder to exploit a system;

5) Fail Secure – limit amount of information exposed on errors encountered by a system;

6) Economy of Mechanisms – keep things simple;

7) Complete Mediation – access to all resources of a system are always validated;

8) Open Design – security based on proven open standards;

9) Psychological Acceptability – security implementation should protect a system but not hamper users of the system;

10) Weakest Link – any system is only as strong as its weakest link;

11) Single Point of Failure – consider adding redundancy to critical systems.

In the process of developing software for modeling the various characteristics of software architectures, there are a number of notations and methods. UML and related technologies provide a popular approach to modeling at the present time. The modeling describes the architecture from the point of view of various stakeholders and their problems.

UML offers expansion mechanisms in the form of notations. They can be either stereotypes or pairs of tag-values. Using profiles you can give specific content to simulated elements marked with these labels. The UMLsec extension to UML is used to use such solutions to express security requirements [13].

UMLsec consists of the following chart types that describe various system representations: a usage diagram, exercise diagram, class diagram, sequence diagram, state chart diagram, and deployment diagram .

Here is a combination of process-oriented, prioritized use with a goal-oriented approach. More specifically, it is necessary to develop a tree security goals along with the development of a system specification. Safety objectives are improved in parallel, giving more system details at the next stages of design. The process is generally iterative. In order not to complicate the examples, relatively simple target trees are used. For instance, Fig. 4.4 discusses the Internet based on business applications.

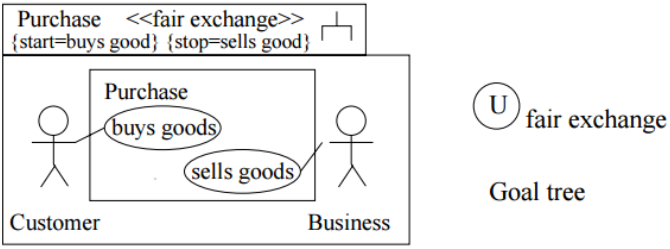


Fig. 4.4. Use case diagram with goal tree

Since the formal UML fragment will be used, one can reasonably argue, for example, that this system is protected in exactly the same way as its individual components are protected. This way, you can reduce the security of the general system to the safety of the mechanisms used (for example, security protocols). The purpose is to show the conditions under which protocols can be used safely in the context of the system.

A purpose-oriented approach to requirements can work better for non-functional requirements than the approach of use cases. However, some studies point to the fact that the goal-oriented analysis and object-oriented analysis complement each other. Thus, it is possible to combine a tree of approaches to nonfunctional requirements using UML. In particular, it is proposed to combine an approach that focuses on options for use for functional requirements with an approach geared towards security requirements. It takes into account the fact that security requirements (for example, privacy) are often applied to certain system

functions, and not to the system as a whole, since application of the security requirement for the whole system may be impracticable.

4.3 Secure Coding. Testing for Security

Depending on the software development process or design approach used, as well as the design and development, coding and testing schedules, it can be performed several times, can be performed at different times for different parts of the software, or can be performed simultaneously with the actions of other phases. At the stage of software coding, the following security issues should be considered:

– selecting the programing language;

– selecting the compiler, library, and runtime;

– agreements and coding rules;

– comments;

– documentation of security-sensitive codes, designs and implementation solutions;

– integration of non-development software;

– required for filters and wrappers.

Practically in all software methods testing is integrated into the coding phase with minimal debugging and modular testing.

A lot of information on specific methods of writing secure code are published. Some of them are organized in a language or platform. Many of them are aimed at a mass audience and do not assume any knowledge about the creation of software. Those principles that are cited by most studies and that are in fact secure coding and secure architecture / design are presented, for example, in [14].

The secure coding standards offered by the CERT CMU are based on documented versions in the standard language. To date, CERT has published secure encoding standards for C (ISO / IEC 9899: 1999 / Cor 3: 2007) and C ++ (ISO / IEC 9899: 1999 / Cor 3: 2007), with plans to publish additional standards for the Sun Microsystems API for Java2 platform Standard Edition 5.0 Specification and programming language Microsoft C # (ISO / IEC 23270: 2006).

Security analysis and software testing is performed regardless of the kinds of software functionality. Its function is to evaluate the security properties of this software when interacting with external objects, such as users, the user environment, other software, and interacting

with each other. The main purpose of software security analysis and testing is to verify that the software has the following properties [14]:

– its behavior is predictable and safe and does not contain vulnerabilities;

– its error handling and exception handling procedures allow you to maintain safe when encountering attack patterns or intentional errors;

– it meets the requirements of non-functional security;

– the source code contains mechanisms for opposing reverse engineering.

A range of security reviews, analyses, and tests can be mapped to the different software life cycle phases (Fig. 4.5).

Requirements	**Security review of requirements and abuse/misuse cases**
Architecture/Product Design	Architectural risk analysis (including external reviews)
Detailed Design	Security review of design. Development of test plans, including security tests.
Coding/Unit Testing	Code review (static and dynamic analysis), white box testing
Assembly/Integration Testing	Black box testing (fault injection, fuzz testing)
System Testing	Black box testing, vulnerability scanning
Distribution/ Deployment	Penetration testing (by software testing expert), vulnerability scanning, impact analysis of patches
Maintenance/support	(Feedback loop into previous phases), impact analysis of patches and updates

Fig. 4.5. Security Reviews and Tests throughout the SDLC

One of the key aspects of software management is the security of any software supplied by the organization. At the same time, insuffi-

cient attention is paid to testing program security and fixing vulnerabilities at earlier stages of the development lifecycle, without waiting for testing for vulnerabilities of the finished application. Security code audit is a structural testing of software to identify vulnerabilities, the implementation of which can reduce the level of integrity, availability and confidentiality of the system. An important feature of security code security technologies is that the main task of the audit is to identify not all possible vulnerabilities, but only code vulnerabilities that can be exploited by an attacker.

In general terms, code security auditing is an iterative process, including planning, analysis, recommendations for finalizing the program and documentation, and developing methods and tools for identifying and analyzing vulnerabilities [15].

We list the main classes of these vulnerabilities:

– overflow, read and write outside the buffer;

– the output of calculations beyond a certain range when converting variables of a numerical type;

– the formation of a negative value for the length of a string of bytes or the number of elements in the array;

– incorrect casting;

– lack of initialization of data;

– leak, shortage, use of freed memory;

– time and synchronization errors;

– errors of locks in multithreaded environments, etc.

These vulnerability classes can be used to perform denial of service attacks or execute illegitimate code.

There are several methods for auditing code security:

– viewing (inspecting) the code manually;

– static code analysis by template;

– dynamic analysis of code execution.

The first approach is considered the most effective in terms of completeness and accuracy of the checks. The shortcomings of the method include high labor intensity and requirements for qualifications and experience of experts. Static code analysis using a template consists in the use of automation tools for searching and analyzing potentially dangerous code constructs (signatures) in the source code of programs. This method is effective when searching for simple vulnerabilities and non-maskable bookmarks, such as buffer overflow, password constants

or "logical bombs". The automated methods for carrying out the static method by template include vulnerability scanners for PREfix, PREfast, AK-Sun, UCA, FlawFinder, ITS4, RATS, FxCop.

Modern code scanners allow to some extent automate:

– search for buffer overflow vulnerabilities;

– search for OS-injections (execution of arbitrary commands);

– SQL injection search;

– search for XSS queries (crossite scripting);

– search for errors in input and output values;

– carrying out a structural analysis of subprograms that implement protection functions.

Numerous studies have shown that the cost of fixing vulnerability after the end of development can be several hundred times higher than the cost of solving the problem in an application that is still being developed.

A penetration test (or pentest) is a practical way to show how much software is protected from threats to information security [16]. There is also the term ethical hacking.

This method simulates a set of "hacker" attacks, the purpose of which is to penetrate the company's internal network infrastructure, steal and / or modify confidential data, and disrupt critical business processes.

Unlike real intruders, the team of testers observes certain ethical rules in carrying out all works: any dangerous actions are committed only by prior agreement with the customer; the entire scanning process is transparent and planned.

Conducting testing for the penetration of software systems requires professionals, in-depth knowledge of IT security, the ability to think outside the box, apply social engineering techniques, collect and analyze information, as well as creativity.

There are both open and commercial methodologies for conducting penetration tests that are capable of ensuring a guaranteed quality of the service when the whole process is observed. However, in practice the use of only one methodology is not advisable, as a rule, they are used modularly with the necessary revision.

The most popular methodologists of ethical hacking:

– Information Systems Security Assessment Framework (OISSG);

– The Open Source Security Methodology Manual (OSSTMM);

– NIST SP800-115 Technical Guide to Information Security Testing and Assessment;

– ISACA Switzerland - Testing IT Systems Security With Tiger Teams;

– The Information Systems Security Assessment Framework (ISSAF);

– OWASP Testing Methodology;

– BSI - Study A Penetration Testing Model;

– Penetration Test Framework (PTF);

Almost all methodologies provide the following scenario of penetration testing (Fig. 4.6):

– planning a penetration test;

– collect information about target systems;

– vulnerability scan;

– penetration into the system;

– writing and reporting;

– cleaning systems from the effects of the test.

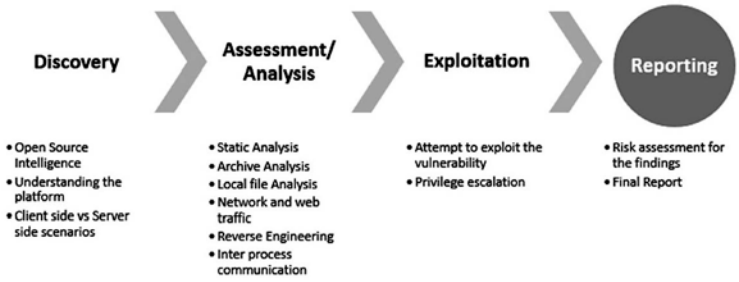


Fig. 4.6. Typical scenario of penetration testing

The penetration test is a complex project and can include several types of work, but at the outset, you should determine the approach to the test: the so-called white box, black box or gray box.

It is important to note that any tool produces a certain number of false positives, such as detecting a false vulnerability or vice versa skipping vulnerability when it really exists.

Since new vulnerabilities in software and technologies are detected almost daily, the penetration test is a complex and voluminous service that can show the current picture of the security of software products.

4.4 Implementing security and resilience into software

When the development of secure software is intended, the overall structure of the technological process is preserved, but at each step additional security measures are taken. Thus, the process of developing secure software is a set of measures that are aimed at providing the required level of information security developed by the software product. There are many methods for developing secure software. The methodologies that consider the construction of secure software involve resources that address particular phases or focus on specific platforms. Most of them are introduced by the largest software developers and various government organizations. The choice of methodology depends entirely on the desired result.

The development of secure software is still closely related to recommendations, best practices and undocumented expertise. The approaches of modern practices provide guidance for specific areas, such as threat modeling, risk management or secure coding. It is important that they be integrated into a comprehensive development method. Despite some improvements in the definition of processes for the development of secure software, these processes are conditioned by the experience of experts. Therefore, it is difficult for developers to evaluate them, assess their strengths and resist their weaknesses. An objective comparison of these methodologies is still an urgent task, and it is difficult for various stakeholders to make a measured decision about which one is more suitable for work.

Next, consider several "ready-to-use" SDLC-methodologies with appropriate information resources that are designed specifically to create secure software. A basic of characteristics will be discussed in order to describe their overall meaning.

Comprehensive, Lightweight Application Security Process (CLASP) [17]. CLASP is the process of creating secure software and contains a set of 24 activities and additional resources that need to be adapted to the development process in use.

Features of the methodology:

– the main goal of CLASP is to support the creation of software primarily from a security perspective;

– CLASP is defined as a set of independent actions that must be integrated into the development process and its working environment. However, this is not an integrated process. The frequency of implementation of activities is indicated on individual activities, which leads to complex coordination. Two road maps give some recommendations on how to combine actions into a single and ordered set;

– CLASP defines roles and assigns actions to these roles. Roles can influence the security situation and are used as an additional perspective for structuring a set of actions;

– CLASP provides a set of security resources, one of which is a list of 104 known types of problems. They determine the basis of security vulnerabilities in the source code of the application.

Security Development Lifecycle (SDL) [18].

SDL was created by Microsoft to solve security problems and includes a set of activities that complement the development process of Microsoft and is designed as an addition to the process of creating software.

Features of the methodology:

– SDL's main goal is to improve the quality of software-oriented software. Most often, this activity is related to the development on the basis of functionality. Because the architecture can primarily reduce security threats, threat modeling begins with architectural dependencies with external systems. However, little attention is paid to the implementation and integration of security mechanisms in the methodology;

– the SDL process is well organized, and related activities are grouped in stages. Specific security stages are comparable with the standard stages of software development, and some activities are continuous in the SDL process (for example, threat modeling and education). Support for revising and improving intermediate results is also present in the SDL process;

– SDL well describes the methods for performing actions, which on average are specific and pragmatic. In particular, the use of flowcharts reduces the attack surface, and threat modeling is described as a more detailed process. Thus, performing some operations is possible even for less experienced developers.

TSP-Secure [19, 20].

CMU's SEI and CERT Coordination Center (CERT/CC) developed the Team Software Process for Secure Software Development (TSP-Secure). The purpose of creating TSP-Secure was to reduce or eliminate software vulnerabilities that arise from design and software errors. An important problem is also the ability to predict the likelihood of vulnerability in the software. In doing so, TSP provides a systematic way for software developers and managers to learn how to implement methods in an organization.

TSP-Secure implements security practices throughout the SDLC and provides approaches and methods for:

– establishment of operational procedures, organizational policies, management oversight, resource allocation, training, planning and project tracking;

– analysis of vulnerabilities by type of defects;

– establishment of safety-related forecasting indicators, control points and safety-related measurements;

– risk management and feedback;

– secure design, use of design patterns to avoid common vulnerabilities and develop security reviews;

– quality management for secure programming, the use of secure language subsets and coding standards, static and dynamic analysis tools;

– security checks, which include the development of plans for various types of security testing;

– remove vulnerabilities from the software.

Secure Software Engineering (S2e) [21].

SSDM is a unified model that combines some of the existing security technologies with the software development process.

The process of creating secure software is divided into five stages:

– security training. The essence of security training is to provide training in security matters;

– threat modeling. Since common security criteria may not be suitable for all software products, each software development must have its own threat model;

– security specification (SS). This implies directing directions and procedures that guarantee the security of the software;

– SS review. Assumes checking the compliance of software design content;

– penetration testing. The capabilities of the software to prevent attacks are tested.

Secure Tropos [22].

Tropos2 is a software development methodology focused on the agent, designed to describe the organizational environment of the multi-agent system and the system itself. Three key aspects:

– consider all stages of system development, adopting a uniform and homogeneous method based on the concept of agents: subjects, goals, objectives, resources and intentional dependencies;

– Tropos pays great attention to early requirements, and also how the proposed system will meet its organizational goals;

– the methodology is based on the idea of constructing a model of the system, which is gradually refined and expanded from the conceptual level to the executed artifacts. This allows developers to accurately check the development process, detailing the higher-level representations presented in the previous development stages.

Using the Tropos2 methodology to address security and resilience issues at all stages of software development, the following three objectives should be considered:

– determine the requirements for system security;

– develop a project that meets the specified safety requirements;

– to approve the developed system in respect of safety.

Thus, Secure Tropos provides a well-managed process that provides developers with security concerns through various modeling activities. The use of the same concepts and designations at all stages of development acts as a key point in the methodology of Secure Tropos.

Undoubtedly, there are more research results on the development of secure software than it was presented above, for example, [10, 14].

Conclusions

Now that you have completed this webquest on Computer Security you are now aware of the possible security treats to computer systems. Not only that, but you are now better able to protect your computers as well as recommend security measures to others.

Questions for self-control

1. When designing a software architecture, which quality attribute do you value more: flexibility or resilience?
2. What is difference between functional and non-functional requirements in software designing?
3. What is software resilience?
4. What are the Software Development Life Cycle (SDLC) phases?
5. Why is it important to create a threat model for each software product separately?
6. Describe the main features Secure Software.
7. According to SSDM what are the phases of creating secure software?
8. What is the main goal of CLASP?
9. What is the purpose of creating TSP-Secure?
10. Explain why does security is a non-functional requirements.
11. What tool is using due to security requirements and test case generation?
12. What can the result of buffer overflows exploitation?
13. What are UMLSec notation diagrams?
14. What is secure design pattern?

References

1. Avizienis. A., Laprie. J.-C.. Randell, B.. Landwehr. C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans, on Dependable and Secure Computing 1. 11-33 (2004).
2. The Difference Between Reliable and Resilient Software. Accessed at: http://cabforward.com/the-difference-between-reliable-and-resilient-software/.
3. Trivedi, K.S., Kim. D.S.. Ghosh. R.: Resilience in computer systems and networks. In: Proc. of Int. Conf. on Computer-Aided Design - ICCAD 2009. p. 74. ACM Press, New York (2009).
4. Laprie. J.-C.: From dependability to resilience. In: 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Fast Abstracts (2008).
5. Gorbenko, A., Kharchenko, V., Mamutov, S., Tarasyuk. O., Romanovsky, A.: Exploring Uncertainty of Delays as a Factor in End-

to-End Cloud Response Time. In: 2012 Ninth European Dependable Computing Conference, pp. 185-190. IEEE (2012).

6. Security in the SDLC process. Accessed at: https://www.owasp.org/index.php?title=File:S-SDLC_TP.jpg&setlang=en.

7. Merkow, Mark S. Secure and resilient software development / Mark S. Merkow, Lakshmikanth Raghavan.

8. Data Breaches Increase 40 Percent in 2016, Finds New Report from Identity Theft Resource Center and CyberScout. Accessed at: http://www.idtheftcenter.org/2016databreaches.html.

9. Model-based evaluation of the resilience of critical infrastructures under cyber attacks

10. Karen Mercedes Goertzel, Allen Hamilton. Enhancing the Development Life Cycle to Produce Secure Software. Accessed at: https://www.researchgate.net/publication/228704603_Enhancing_the_Development_Life_Cycle_to_Produce_Secure_Software.

11. Chad Dougherty, Kirk Sayre, Robert C. Seacord, David Svoboda, Kazuya Togashi. Secure Design Patterns. Accessed at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9115.

12. Samuel T. Redwine, Jr., ed., Secure Software: a Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software. Available from: https://buildsecurityin.us-cert.gov/daisy/bsi/resources/dhs/95.html?branch=1&language=1.

13. Jan JürjensSiv Hilde Houmb. Risk-Driven Development of Security-Critical Systems Using UMLsec. Accessed at: https://link.springer.com/chapter/10.1007/1-4020-8159-6_2.

14. Software Security Assurance State-of-the-Art Report (SOAR). Accessed at: https://www.csiac.org/wp-content/uploads/2016/02/security.pdf.

15. Security Code Review in the SDLC. Accessed at: https://www.owasp.org/index.php/Security_Code_Review_in_the_SDLC.

16. Ric Messier. Penetration Testing Basics: A Quick-Start Guide to Breaking into Systems / Apress, 2016. – 115 p.

17. Open Web Application Security Project (OWASP) CLASP Project Webpage. Accessed at: https://www.owasp.org/index.php/CLASP_Concepts.

18. Lipner, Steve and Michael Howard. “The Trustworthy Computing Security Development Lifecycle”. Microsoft Developer Network,

March 2005. Accessed at: https://msdn.microsoft.com/uk-ua/enus/library/ms995349.aspx.

19. Over, James W. “TSP for Secure Systems Development” (presentation). Accessed at: https://www.sei.cmu.edu/tsp/tsp-securepresentation/tsp-secure.pdf.

20. Schneider, Thorsten. “Secure Software Engineering Processes: Improving the Software Development Life Cycle to Combat Vulnerability”. Software Quality Professional. Volume 8 Issue 1, December 2006. Available from: https://secure.asq.org/perl/msg.pl?prvurl=http://asq.org/pub/sqp/past/vol9_issue1/sqpv9i1schneider.pdf.

21. A. S. Sodiya, S. A. Onashoga, and O. B. Ajayi. Towards Building Secure Software Sysems. Accessed at: https://www.researchgate.net/publication/249724183_Towards_Building_Secure_Software_Systems.

22. Mouratidis, Haralambos and Paolo Giorgini (2007) “Secure Tropos: A Security-Oriented Extension of the Tropos Methodology”. International Journal of Software Engineering and Knowledge Engineering, Volume 17 No. 2, April 2007, pages 285-309. Accessed 25 August 2008 at: http://www.dit.unitn.it/~pgiorgio/papers/IJSEKE06-1.pdf

АННОТАЦИЯ

В разделе актуализирована проблема информационной безопасности программных продуктов и освещено понятие резильентного программного обеспечения.

Раздел содержит описания практик безопасного проектирования приложений и нотации UMLsec, которые используются для создания защищенных программных систем.

В разделе рассмотрено типовые уязвимости программного обеспечения, техники безопасного кодирования, приёмы динамического и статического анализа кода, обосновано важность проведения тестирования на проникновение.

Также раздел содержит описание ключевых особенностей наиболее популярных методологий создания безопасного программного обеспечения.

У розділі актуалізовано проблему інформаційної безпеки програмних продуктів та висвітлено поняття резільєнтного програмного забезпечення.

Розділ містить опис практик безпечного проектування програм і нотацій UMLsec, які використовуються для створення захищених програмних систем.

У розділі розглянуто типові вразливості програмного забезпечення, техніки безпечного кодування, прийоми динамічного і статичного аналізу коду, обґрунтовано важливість проведення тестування на проникнення.

Також розділ містить опис ключових особливостей найбільш популярних методологій створення безпечного програмного забезпечення.

In the section the problem of information security of software products is actualized and the concept of resilient software is covered.

The section contains descriptions of security application design practices and UMLsec notations that are used to create secure software systems.

The section describes typical software vulnerabilities, secure encoding techniques, dynamic and static code analysis techniques, the importance of conducting penetration testing is justified.

The section also describes the key features of the most popular methodologies for secure software development.

PART 2. FUNDAMENTALS OF FORMAL METHODS FOR SYSTEM SECURITY

GLOSSARY

API – Application Programming Interface
AS – Autonomous system
DES - Data Encryption Standard
DSA - Digital Signature Algorithm
DFS - depth first search
ICTS - Information and Communication Technological systems
BPMN - Business Process Model and Notation
DREF -Dependability and Resilience Engineering Framework
DSL - domain specific modelling languages
SPL - software product line
CSP - Communicating sequential processes
RF - REACT framework
CCS - Calculus of Communicating Systems
CC - Common Criteria
EAL - Evaluation Assessment Level
CICS - Customer Information Control System
OOA - object-oriented analysis
VDM - Vienna Development Method
OSSD - Open Source Software Development
SUMO - Upper Merged Ontology
TAF - Test Automation Framework
SCR - Software Cost Reduction
XML - Extensible Markup Language
DoS - denial of service
CICS - Customer Information Control System
GUI - Graphical user interface
UCM - Use Case Maps
GRL - GALS Representation Language
UCM - Use Case Maps
FDAF - Formal Design Analysis Framework

GSE - Genetic Software Engineering

R2D2C - Requirements to Design to Code

HR - Human Resources

STC - short term contracts

OCL - Object Constraint Language

QCO - Quality Control Officer

DNS - Domain Name System

KAOS - Knowledge Acquisition for autOmated Specifications

UML - Unified Modeling Language

AADL - Architecture Analysis and Design Language

CERT/CC - Computer Emergency Readiness Team Coordination Center

OCL - Object Constraint Language

PKI - public-key infrastructure

MAC - Mandatory Access Control

SSL - secure sockets layer

RBAC - role-based access control model

MCA - multiple certification-authority

KDM - key-dependent message

OTP - One-Time Pad

EJB - Enterprise JavaBeans

SMASC - Separating Modelling of Application and Security Concerns

SAM - Software Architecture Model

COI - Conflict of interest classes

Multilevel Security (MLS)

DARPA -Defense Advanced Research Projects Agency

PCA - Polymorphous Computing Architecture

AAP - Avionics Application Process

EPE - Encryption Processing Element

NSE - Network Security Element

DAC - Discretionary Access Control

DTLTS - discrete-time transition labeled transition system

CVS - Concurrent Versions System
AOSD - Oriented Software Development
PIM - Personal Information Management
ED - Embedded Device
FADSE - Formal Analysis and Design for Security Engineering
ID – Identifier
IEEE – Institute of Electrical and Electronics Engineers
IP – Internet Protocol
JFK - Just Fast Keying
GA - General Attacker
LAN – Local Area Network
MAS - multi-agent systems

CHAPTER 5. INTRODUCTION TO FORMAL METHODS

Content of the CHAPTER 5

5.1 What are Formal Methods?

The term “formal methods” has come into common use (and abuse) during the past years. In this book we take a fairly liberal interpretation of the term. We include, for example, not only “mainstream” formal methods such as Event-B, Z, VDM, CSP and CCS, but also other programming and system design paradigms which are underpinned by discrete mathematics, for example code generation and transformation, techniques and tools for static analysis of programs, and programming languages with sound semantics [1-7].

A software specification and production method, based on a mathematical system, that comprises: a collection of mathematical notations addressing the specification, design and development phases of software production; a well-founded logical system in which formal verification and proofs of other properties can be formulated; and methodological framework within which software may be verified from the specification in a formally verifiable manner.

This is a rather ambitious definition. Formal methods are attractive, but in practice most them in common use do not address the full spectrum of design, some supporting specification phases, some the construction phases, and others the analysis of systems.

Typically, formal methods have three components:

- a notion of “program”;
- a means of expressing properties of the computation of programs;
- some method for establishing whether a program has some property.

Each of these is rigorously mathematically defined. This is a fairly liberal description of a formal method. It covers things like the type inference system of some modern programming languages which “prove” programs have a well-formed type, to entirely general purpose theorem proving systems or proof assistants, or paper and pencil methods. The quality of evidence one gets from a formal method varies according to the method but generally it is highly accurate, is convincing to trained workers, and often has a rather narrow coverage because it abstracts from many detailed features of systems.

There is also a wide range of formality and rigour applied in the use of formal methods. A *formal proof* might consist of justifying a conjecture by deriving it from the basic axioms of the mathematics upon which the logical system in use is based. A *rigorous argument* looks more like an outline of how a proof might proceed, but would not supply all the intervening detail. Additionally, rigorous argument may draw upon a rich body of known results, but without the need for formally integrating them into a proof. Note that this is how engineers and mathematicians usually work—they customarily have a commonly understood context which obviates the need to descend into detail which obscures the main subject. However computer based proof tools do not have the insight of mathematicians, and cannot interpolate unstated detail between steps. Computer based proofs are therefore usually very detailed, and can be extremely obscure, although their validity is potentially less contentious than a rigorous argument [8-16].

The mathematical basis of formal methods may be an existing part of mathematics—for example the Z specification language has set theory at its heart—or can be developed anew for the method—the Calculus of Communicating Systems has a theory presented in an algebraic style, but specific to the Calculus.

There is a large and growing variety of formal methods, of varying age and maturity. It must be realised that there is no "standard" formal method. Each technique has particular strengths and weaknesses and it may be that in the course of systems development it is appropriate to use a number of methods at different stages of the process. Just as in traditional engineering, no single theory encompasses all aspects of design and development. Many theoretical approaches are used—either explicitly or implicitly. "Stock answers" are often brought to bear and, even if these appear not to involve mathematics directly, they usually have a long history of mathematical analysis which has consolidated into a precise understanding of the nature and behaviour of a particular aspect of the component or structure. The interaction and compositional properties of such components are well understood and the properties of the whole (strength, weight, current, speed, loading, cost, time to construct, etc.) can all be predicted with much more accuracy than is now possible with comparable computer systems development.

The need for and impact of this use of theory is also well understood by managers, engineers' licensors and procurers. Comparable use of methods or theories is currently not so diverse in the area of computer systems engineering and it is desirable that engineers cultivate a broad awareness of possible methods which are relevant to different aspect of systems design and develop the ability to assess the usefulness and applicability for particular applications of competing or complementary approaches.

Just as there is no standard formal method, there is (as yet) no standard way of applying any particular formal method. As case study material and tools are developed to support the engineering domain, this may change. However until then it is likely that engineering application of formal methods will require to be open in nature with the partial success or failure of certain approaches expected and the results of this failure used to inform more successful application.

As with other engineering disciplines, a body of "good practice" should provide guidance to engineers as to effective use of techniques within different contexts. Unfortunately with formal methods this material is sparse, of variable quality, and not well indexed. In due course this body of information will grow, in particular it is hoped that a range of texts and real-world case studies will be produced.

Let us go on to present the capabilities and limitations of formal methods, and guidance for their uptake.

5.2 The Nature of Formal Methods

Software and systems engineering have a number of different life cycle models, however most break down into a number of phases of activity such as: requirements, specification, design, implementation and test [17-23].

A common view of the use of formal methods in this regime might expect the following. The customer interacts with the supplier closely over requirements. The supplier writes a formal specification, which is then successively refined as implementation detail is added until an implementation results. This is then subjected to test. The use of formal methods at the various stages of refinement can mean that the final implementation in some sense is proven to satisfy the top-level specification. The 'proof of satisfaction' is one of the documents which

may be required by the certification authority as part of any licensing procedure.

Traditionally this kind of approach has been seen as the goal of formal methods research and there is still considerable academic investigation into approaches which aim to achieve this. However the level of "correctness" delivered must be interpreted within a broad engineering framework—the meaning of the word to the customer may be very different to that assumed by the designer. Absolute correctness is unattainable—it is not a concept that is familiar to (or sought by) engineers in other disciplines, they do however make extensive use of mathematics to model, design and analyse.

In this hierarchy the requirement and specification layer generally say mostly *what* the system is required to do. The lower layers (and there is usually more than just one step at each level) contain increasing amounts of implementation detail *how* the system is to achieve its function. Correspondingly the amount of information at each level increases as we descend, usually very rapidly.

Let us comment on and elaborate this model. There are several observations to be made. Note that many of these are not necessarily restricted to formal methods, but are often true of software engineering in general.

Requirements are never right

The symptom of the supplier delivering what he was asked for and the customer only then realising that the original requirements were wrong or ambiguous is common. This is one of the major problems in systems and software engineering. Close interaction between the customer and the supplier is desirable at all stages of development, but identifying problems as early as possible in the life-cycle is highly desirable. Use of formal methods at the requirements stage can increase the clarity of understanding and so reduce the scope for misinterpretation with the corresponding saving of wasted development effort, saving both time and money.

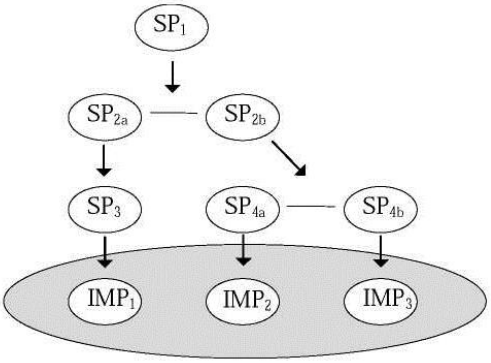
One approach is to use a formal method that supports executable specifications. This does indeed allow experimentation with statements of requirements and can result in a significant "feel good" factor for the customer as they have something concrete (i.e. runable) in their hands early on. However, this can mean that implementation level decisions

(how rather than what) are made too early. Inappropriate use of such languages can restrict freedom at the design stage and can remove the possibility of simplification or generalisation of the system. It may be preferable to develop a prototype than to use an executable specification language. However any prototype should be developed from the point of view of informing the requirements analysis rather than guiding the implementation.

It is important that formal languages for requirements or specification are used in the appropriate style and not as programming languages. Avoiding this trap can be difficult if one is moving existing programming staff into the use of formal specification languages.

Real world problems are complicated

In some cases it is possible nowadays to "verify" code against specifications for some systems of perhaps up to 20,000 lines of code. However most systems nowadays are much bigger than this and to apply the above paradigm to large systems we need to support the idea of decomposition as well as refinement (Fig.5.1).



Here a top level specification (SP_1) is decomposed and refined into two subsidiary specifications (SP_2 and SP_3). Note that there are two kinds of relationship here. Horizontal lines are a glue that binds components together and the vertical connection represents the refinement we have talked about already. Thus each level is guaranteed to implement the specification it "descends" from. Ultimately we end up with a number of executable modules (IMP_{1-3}) which implement SP_1.

Figure 5.1. Decomposition and refinement of the specification

In this model the implementations delivered at the bottom are linked formally only at levels above this. This means that we need an integrated theory of refinement and decomposition as well as an implementation language which supports the communication behaviour of these "modules" with respect to the higher level specification of decomposition. In practice this means the semantics of both the

specification language and the implementation language must be related formally so that the behaviour of the implementation is adequately proven to satisfy the higher level specification.

An alternative approach is to use a single "wide spectrum" language. These are languages which support both specification and implementation activities. The term "wide spectrum" is used to refer to the spectrum of activities in system development. In some cases they provide mechanisms for verifying the code against specification, or support formal refinement of code from specifications. Although these exist in prototype form this area is still the subject of much research.

One benefit of the approach in figure 5.1 is the separation of concerns possible under this model. It may be possible to isolate critical aspects of system behaviour in a small number of components. The use of formal methods to ensure integrity of these components can then be more focused. Traditional techniques may then be adequate for the remainder, although great care must be taken over interfaces.

Formality is internal to the model above

Even if the formal proof of correctness of implementation against specification exists this still only represents a part of the final system. The implementation will be expressed in some high level programming language. This will require to be compiled to object code, probably with the linking in of library components. This then runs on some hardware. It is possible to verify components such as compilers, library function and hardware, but these would generally use different techniques and to reason about the "correctness" of the composition of all of these is a considerable task requiring significant intellectual and economic investment. These kinds of activities usually require strategic sponsorship, and involvement of academic groups, although we can expect results of this work to become viable (technically and economically) in the industrial context during the next decade [24-28].

Other problems arise in that the resulting system has to interface with the environment which includes unpredictable entities like people or analogue components often in an asynchronous environment. Here the main problem is one of *validation* of the system with respect to its environment. Here again formal methods can help. One can model interfaces, and use these to explore the operating conditions of the plant, thus exposing the requirements on the operator, as well as the

system. One can exercise a formal specification of a system with respect to an environmental model such as a fault tree analysis (which itself may be expressed in a formal language).

Even if code is verified with respect to a specification, it must then be compiled to execute on target platforms, nowadays the simplest of which have executives, and increasingly, full blown operating systems which support the application code. Nowadays these are intricate and highly specialised, and generally not amenable to formal verification. Unlike application code, however, compilers and operating systems may have a significant history of successful operation, and this can often be used to justify their use. Ultimately the system is executed on physical components (wire, glass, silicon etc.) which (at least in this context) are also analogue in nature. The behaviour of all of this combined is certainly impossible to model formally so one must ultimately call a halt to the process of full verification and decide what concrete benefit the application of formal methods can actually give.

In practice one must choose from a range of approaches and it worth extending the model above with the idea of property oriented specification (Figure 5.2).

Typically these properties will state some aspect of dependability required of the system, such as safety, security, reliability and performance. In a fully formal model of this kind one would expect properties for lower levels of specification to include higher level properties plus some more which are relevant to that level of implementation. For example, one would expect performance related properties only to be relevant for verification and validation purposes at the lower levels of specification or implementation [29-33].

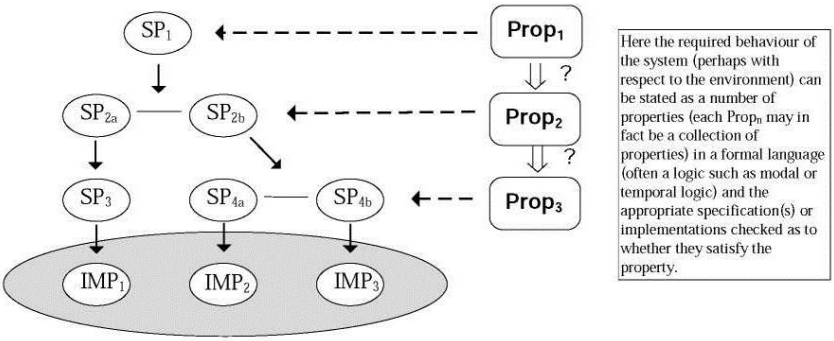


Fig 5.2 Decomposition and refinement of the property oriented specification

Such approaches may require the replacement of much of existing systems engineering practice.

It may be appropriate for a limited number of systems (or clearly delineated components) which are small enough and critical enough to consider "full" verification in conjunction with good testing practice. For most applications however this will not be appropriate. It would require a major change to systems engineering practice. Formal methods are by and large based upon technological approaches, whereas the successful management of the development of computer systems is as much a social problem. Much of the systems engineering research in the past two decades which has successfully been deployed in industry is focused around the management of people, activities and communication. "Traditional" formal methods do not fit well with these approaches with their strong focus on the technical object being produced (the program) rather than the teams doing the production (the programmers, designers, testers etc.).

Application of formal methods *is* possible in ways which will integrate and provide complementary strengths to existing methods. By using this approach it is possible to evolve existing practice to integrate these new approaches. It also handles the risk problem well in that the

consequences of failure are localised and can usually be mitigated by reverting to already well known and understood good practice.

5.3 Benefits in the use of Formal Methods

As described above formal methods can be used to provide a high degree of assurance of a system's correctness with respect to a specification. While in some applications the assurance that " proof of correctness" itself is the main goal, in many other applications the primary benefit of formal approaches to system design is not the "proof", but the increased understanding of the problem which the *process* of proving or formally specifying provides, and the increase in confidence in and support for the design process (and final product) [34-40].

In addition to the benefits of formal specification or proof of correctness there are many other ways in which use of formal approaches can be beneficial. Note that these are not achievable with every formal method, or with each application, but the benefits can include:

To raise engineering understanding of the problem or application

The process of thinking formally about a system will almost certainly force clearer thinking, and hence understanding, about the nature and purpose of the system being built. This benefit alone can justify the use of formal methods.

To raise customer confidence in the product

Delivering a faulty product can do enormous damage to your customer base. While formal methods are unlikely to remove all errors, they can be used to analyse critical aspect of system behaviour, and potentially eliminate specific classes of errors.

To raise confidence in the development method

This then supports the supplier in making clearer claims about the quality/cost/development time and effort of a product.

To document the design process unambiguously

This can help all those involved: the project manager, designer/engineer, certifier/validation and verification team, maintainer, customer.

To reduce maintenance effort and cost

Much of the life-cycle cost of current systems is not in the original design, but in system support, modification and maintenance. The clear documentation produced when formal methods are used results in significant savings in this phase of the life-cycle. The cost of rectifying a fault once a system is in operation is orders of magnitude greater than correcting it at the design stage. In addition, some formal methods provide active support during maintenance stages by identifying the scope of alterations and thus limiting the retesting required of the modified system.

To identify mistakes and omissions early

Formal approaches tend to detect problems early in the design cycle when they are therefore much easier and cheaper to correct.

To shorten the time to market

Despite the fact that more effort needs to be invested in the specification phases when no deliverable code is produced, the resulting system should have fewer errors. Having to reengineer a product after being released and found to be faulty can cost enormous amounts in lost market share. Experience has shown that it is straightforward to produce code quickly and efficiently from formally produced specifications. Significant gains in programming team productivity is therefore possible.

To market your product/service/company

An ability with formal methods in your organisation may confer a significant competitive advantage in certain markets e.g. security or safety. Indeed in some sectors it may not be possible to tender without ability in formal methods.

To protect yourself/company

Product liability laws may leave you exposed to costly litigation in the event of disaster or non-performance, if you cannot demonstrate that you have adopted the best design and validation techniques available and appropriate.

To aid the certification process

The use of formal methods is increasingly being advised (and in some cases mandated) in standards and guidelines. Their use can therefore be a major aid in convincing a certifier that the system is safe to deploy.

5.4 Limitations to the use of Formal Methods

It is important to realise limitations in the use of formal methods, as well as their benefits. Outlined below are some of the disadvantages or potential problems which may arise and should be taken into account in making a decision on whether to use formal methods.

Formal methods can be expensive to do

Perhaps more accurately—formal methods *will* be expensive the first time you try them.

It is important to amortise start-up costs across a spread of projects and time and to start out with simple and easily achievable aims, ramping up as technical experience grows. This said, the use of experienced consultants in the early stages on use of formal methods will usually be a low risk and cost effective route to adoption.

There is no formulaic approach

Case study material is sparse. This will improve in time and with experience, but you must expect to view projects from an applied *research* perspective rather than established engineering *practice*. Appropriate risk management techniques must be used. The use of appropriately experienced consultants for early ventures in the field can reduce both cost and risk.

Training time

This varies by method, but it takes at least several months for even a well motivated engineer to become familiar with the use of formal methods in design. (Attaining a *reading* capability can be much quicker though).

Difficulties of communications

Using formal methods to communicate can increase accuracy, but does require a change in attitude, and some training. This can be a particular problem with customers and management who do not have the *technical* incentive to become familiar with them. Integrating formal methods into existing development methods, and proper use of interspersed text and engineering diagrams and notations, can mitigate the problem.

The complexity/size problem

Although formal methods are improving their range of application the problem of scaling up to larger examples can be problematic. Abstraction is a common approach in many engineering disciplines for

handling such complexity. In systems engineering however our understanding of approaches to abstraction is not well developed. Again the use of skilled consultants or staff can help here.

Formal methods are weak on performance

Research is improving in this area, for example in the development of timed calculi, and in integrating with more traditional performance approaches, but it will be some time before these mature.

Poor formal infrastructure

While principles of applications of formal methods are well understood, maximum benefit requires adequate formal infrastructure. For example almost all programming languages do not have a *usable* formal semantics. This makes the behavioural analysis of code difficult, even though formal methods might have been used earlier in the development cycle. Similarly no *usable* verified compilers exist, so how can one be sure of the "correctness" of object code.

Cultural uncertainty surrounding formal methods

Formal methods are a moving target. Which ones work in what domain? What is next year's method going to be? Will I have to completely retrain?

The validation problem

Formal methods apply to abstractions. Real systems are too complicated to analyse fully. (This is true in traditional engineering as well though). Can I validate my model with respect to the expected behaviour of the whole system? What level of requirement/specification is appropriate?

Lack of tools

Although the situation is improving tool coverage is still poor, and doing formal methods by hand is time consuming, error prone and usually will require a higher level of expertise. It's a bit like trying to write a program without a compiler. Even simple tools such as syntax and type checking tools can strongly support the use of formal methods.

Formal methods do not substitute for creative thinking

In fact even more is required to bring together traditional design capability with the new approach. It is important that existing expertise and practice are properly utilised with the introduction of formal methods.

Advancement questions

1. What components do formal methods have?
2. What does a formal proof mean?
3. What are the formal methods designed for?
4. What is a rigorous argument in context of the formal methods?
5. What are the main problems in the construction of the formal methods?
6. What are the formal language used for?
7. What is the main idea of the decomposition and refinement of the specification?
8. What is the main feature of the property oriented specification?
9. What are the main benefits in the use of formal methods?
10. What are the main limitations to the use of Formal Methods

REFERENCES

1. Anderson S. O. Guidance on the use of Formal Methods in the Development and Assurance of High Integrity Industrial Computer Systems / S. O. Anderson, R. E. Bloomfield, G. L. Cleland // European workshop on industrial computer systems technical committee 7 (Safety, Reliability and Security). – 1998. – pp.34.
2. Popov P. Application of Formal Methods (Poster Session)-Reliability Assessment of Legacy Safety-Critical Systems Upgraded with Off-the-Shelf Components / P. Popov // Lecture Notes in Computer Science. – 2002. – № 2434. – p. 139-150.
3. Vain J. Refinement-based development of timed systems / J. Vain, J. Berthing, P. Boström, K. Sere, L. Tsiopoulos // International Conference on Integrated Formal Methods. – 2012. – №1. – p. 69-83.
4. Vain J. Model Checking–A New Challenge for Design of Complex Computer-Controlled Systems / J. Vain, R. Kyttner // Proc. of 5-th International Conference on Engineering Design and Automation. – 2001. – №2. – p. 593-598.

5. Vain J. Model checking of emergent behaviour properties of robot swarms / J. Vain, S. Juurik //
6. Proceedings of the Estonian Academy of Sciences. – 2011. – №60. – p. 48-54.
7. Vain J. Model checking response times in Networked Automation Systems using jitter bounds / J. Vain, S. Srinivasan, F. Buonopane, S. Ramaswamy // Computers in Industry. – 2015. – №74. – p. 186-200.
8. Vain J. Formal Techniques for Networked and Distributed Systems-FORTE 2007: 27th IFIP WG 6.1 International Conference, Tallinn, Estonia, June 27-29, 2007, Proceedings / J. Vain // Springer Science & Business Media. – 2007. – №4574. – p. 364-373.
9. Vain J. Model-Based Testing of Real-Time Distributed Systems / J. Vain, E. Halling, G. Kanter, A. Anier, D. Pal // International Baltic Conference on Databases and Information Systems. – 2016. – №2. – p. 272-286.
10. Vain J. Generating optimal test cases for real-time systems using DIVINE model checker / J. Vain, D. Pal // Electronics Conference (BEC), 2016 15th Biennial Baltic. – 2016. – №3. – p. 99-102.
11. Vain J. Design and verification of Cyber-Physical Systems using TrueTime, evolutionary optimization and UPPAAL / J. Vain, S. Balasubramaniyan, S. Srinivasan, F. Buonopane, B. Subathra //
12. Microprocessors and Microsystems. – 2016. – №42. – p. 37-48.
13. Vain J. Model checking in planning resource-sharing based manufacturing / J. Vain, T. Otto // IFAC Proceedings Volumes. – 2006. – №39. – p. 535-540.
14. Vain J. Integrating Refinement-Based Methods for Developing Timed Systems / J. Vain, L. Tsiopoulos, P. Boström // From

Action Systems to Distributed Systems. – 2016. – №4. – p. 171-185.
15. Vain J. An Agent-based Modeling for Price-responsive Demand Simulation / J. Vain, H. Liu // ICEIS. – 2013. – №1065. – p. 436-443.
16. Tagarev T. Defence Management: An Introduction / T. Tagarev, H. Bucur-Marcu, P. Flur . – Geneva : DCAF - Geneva Centre, 2009. – 212.
17. Tagarev T. Introduction to Program-Based Defense Resource Management / T. Tagarev // Connections: The Quarterly Journal. – 2006. – №5. – p. 55-69.
18. Russo S. Defect analysis in mission-critical software systems: a detailed investigation / S. Russo, G. Carrozza, R. Pietrantuono // Software: Evolution and Process. – 2014. – №1699. – p. 22-49.
19. Russo S. How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation / S. Russo, D. Cotroneo, R. Pietrantuono, T. Kishor // Journal of System and software. – 2016. – №113. – p. 27-43.
20. Russo S. RELAI testing: a technique to assess and improve software reliability / S. Russo, D. Cotroneo, R. Pietrantuono // IEEE Transaction on Software Enginering. – 2015. – №42. – p. 452-475.
21. Russo S. Scalable Analytics for IaaS Cloud Availability / S.Russo, R. Ghosh, F. Longo; F. Frattini; K.S. Trivedi // IEEE Transaction on Cloud Computing. – 2014. – №4. – p. 57-70.
22. Kharchenko V. Software Engineering for Resilient Systems / V. Kharchenko, A. Gorbenko, A. Romanovsky, V. Kharchenko. – Berlin : Springer Berlin Heidelberg, 2013. – p. 199.
23. Kharchenko V. Principles of Formal Methods Integration for Development Fault-Tolerant Systems: Event-B and FME (C) /

V. Kharchenko, A. Tarasyuk, A. Gorbenko // MASAUM Journal of Computing.– 2015. – №3. – p. 423-429.
24. Kharchenko V. Cybersecurity Case for FPGA-Based NPP Instrumentation and Control Systems / V. Kharchenko, O. Illiashenko, Y. Broshevan. – USA : 24th International Conference on Nuclear Engineering, 2016.– 10.
25. Kharchenko V. Complexity-based Prediction of Faults Number for Software Modules Ranking Before Testing: Technique and Case Study / V. Kharchenko, S. Yaremchuk // ICT in Education, Research and Industrial Applications. – 2016. – №5. – p. – 427-440.
26. Kharchenko V. Automation of Quantitative Requirements Determination to Software Reliability of Safety Critical NPP I&C systems / V. Kharchenko, B. Volochiy, O. Mulyak, L. Ozirkovskyi // Second International Symposium on Stochastic Models in Reliability. – 2016. – №6. – p. 337-346.
27. Kharchenko V. Assurance Case driven design for software and hardware description language based systems / V. Kharchenko, V. Sklyar // Радіоелектронні і комп'ютерні системи. – 2006. – №5. – p. 85-90.
28. Garavel H. Formal Methods for Safe and Secure Computers Systems / H. Garavel. – Germany : Federal Office for Information Security, 2013. – 362.
29. Tarasyiuk O. Formal method for the development of the critical software / O. Tarasyiuk, A. Gorbenko //
30. Romanovsky A. Deployment of Formal Methods in Industry: the Legacy of the FP7 ICT DEPLOY Integrated Project / A. Romanovsky // Newcastle University, Computing Science Newcastle upon Tyne. – 2012. – №37. – p. 1-4.
31. Laprie, J-C. From dependability to resilience / J-C. Laprie // 38th Annual IEEE/IFIP International Conference on Dependable Systems and networks. – 2008. – №5. – p.1-2.

32. Lirong D. A Survey of Modelling and Analysis Approaches for Architecting Secure Software Systems / D. Lirong, K. Cooper // International Journal of Network Security. – 2007. – №5. – p.187-198
33. Constance L. Applying Formal Methods to a Certifiably Secure Software System / L. Constance, M. Heitmeyer, M. Archer, I. Elizabeth, D. Leonard, D. John // SOFTWARE ENGINEERING. – 2008. –№34. – p.82-98.
34. Pedro Miguel dos Santos Alves Madeira Adão Formal Methods for the Analysis of Security Protocols / Pedro Miguel dos Santos Alves Madeira Adão // PhD diss., INSTITUTO SUPERIOR TÈCNICO. – 2006
35. Almeida J. Rigorous Software Development. An Introduction to Program Verification / J. Almeida, B.Frade, J. Pinto, J. S. Melo de Sousa S. – Berlin : Springer, 2011. – p. 43.
36. Cortier V. Formal Models and Techniques for Analyzing Security Protocols / V. Cortier, S. Kremer.– France : IOS Press Springer, 2011. – 312.
37. Hassan R. Formal Analysis and Design for Engineering Security (FADES) / R. Hassan Abdel-Moneim Mansour. – Blacksburg : Virginia, 2009. – 235.
38. Christianson B. Security Protocols 14th International Workshop / B. Christianson, B. Crispo J. Malcolm M. Roe. – ISSN 0302-9743 ISBN-10 3-642-04903-6 Springer Berlin Heidelberg New York ISBN-13 978-3-642-04903-3 Springer Berlin Heidelberg New York. – 2006. – 287.
39. Laprie J.-C. Resilience for the Scalability of Dependability / J.-C. Laprie // Fourth IEEE International Symposium on Network Computing and Applications IEEE. – 2005. – №1109. – p. 5-6.
40. Tarasyuk A. Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B / A. Tarasyuk, E.

Troubitsyna, L. Laibinis // Lecture Notes in Computer Science Springer. – 2012. –№7321. – p. 237-252.

CHAPTER 6 FORMAL ANALYSIS AND DESIGN FOR SECURITY ENGINEERINGAN AND RESILIENCE

Content of the CHAPTER 6

6.1 Knowledge Acquisition for automated Specifications – Goal-Oriented Requirements of the Security Engineering

Van Lamsweerde in [1, 2, 3] has described Knowledge Acquisition for autOmated Specifications (KAOS) as a general approach for eliciting, analyzing and modeling functional and non-functional requirements of software systems based on first-order temporal logic. Van Lamsweerde in [4, 5] has then extended KAOS to handle security requirements. Knowledge Acquisition for autOmated Specifications is a requirement engineering method concerned with the elicitation of goals to be achieved by the envisioned system, the operationalization of such goals into specifications of services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices, and software [3]. KAOS employs some techniques based on a temporal logic formalization of goals and domain properties with the aim of deriving more realistic, complete, and robust requirements specifications. The key concept in KAOS is to handle exceptions at requirements engineering time and at the goal level, so that more freedom is left for resolving them in a satisfactory way. The KAOS framework supports the whole process of requirements elaboration, from the high level goals to be achieved to the requirements, objects, and operations to be assigned to the various agents in the composite system [6]. The methodology provides a specification language, an elaboration method, metalevel knowledge used for local guidance during method enactment, and tool support [7].

Concepts and Terminology

KAOS specification language provides constructs for capturing various kinds of concepts that appear during requirements elaboration namely goals, constraints, agents, entities, relationships, events, actions, views and scenarios. There is one construct for each type of concept. The types are defined first followed by the constructs for specifying their instances [8].

Objects: an object is a thing of interest in the domain whose instances may evolve from state to state [9]. They can be:

- agents: active objects;
- entities: passive objects;
- events: instantaneous objects;
- relationships: depend on other objects.

Operations: an operation is an input-output relation over objects. Operation applications define state transitions. Operations are characterized by pre/post and trigger conditions [10]. A distinction is made between domain pre/post conditions, which capture the elementary state transitions defined by operation applications in the domain, and required pre/post conditions, which capture additional strengthening to ensure that the requirements are met [11].

Goal: it's an objective for the system. In general, a goal can be AND/OR refined till we obtain a set of goals achievable by some agents by performing operations on some objects [12]. The refinement process generates a refinement directed acyclic graph. AND-refinement links relate a goal to a set of subgoals (called refinement); this means that satisfying all subgoals in the refinement is a sufficient condition for satisfying the goal. OR-refinement links relate a goal to an alternative set of refinements; this means that satisfying one of the refinements is a sufficient condition for satisfying the goal. Goals often conflict with others. Goals concern the objects they refer to [6].

Requisites, requirements and assumptions: the leaves obtained in the goal refinement graph are called requisites. The requisites that are assigned to the software system are called requirements; those assigned to the interacting environment are called assumptions [5].

Domain property: is a property about objects or operations in the environment which holds independently of the software-to-be. Domain properties include physical laws, regulations, constraints imposed by environmental agents, indicative statements of domain knowledge [13]. In KAOS, domain properties are captured by domain invariants attached to objects and by domain pre/post conditions attached to operations [14].

Scenario: is a domain-consistent sequence of state transitions controlled by corresponding agent instances; domain consistency means that the operation associated with a state transition is applied in a state satisfying its domain precondition together with the various domain invariants attached to the corresponding objects, with a resulting state satisfying its domain post condition.

Goals are classified according to the category of requirements they will drive about the agents concerned [15]. Functional goals result in functional requirements. For example, SatisfactionGoals are functional goals concerned with satisfying agent requests; InformationGoals are goals concerned with keeping agents informed about object states. Likewise, nonfunctional goals result in nonfunctional requirements. For example, AccuracyGoals are nonfunctional goals concerned with maintaining the consistency between the state of objects in the environment and the state of their representation in the software; other subcategories include SafetyGoals, SecurityGoals, PerformanceGoals, and so on [10].

Goal refinement ends when every subgoal is realizable by some individual agent assigned to it, that is, expressible in terms of conditions that are monitorable and controllable by the agent. A requirement is a terminal goal under responsibility of an agent in the software-to-be; an expectation is a terminal goal under responsibility of an agent in the environment (unlike requirements, expectations cannot be enforced by the software-to-be) [16].

The Specification Language

Each construct of the KAOS specification language has a two-level generic structure: an outer semantic net layer for declaring a concept, its attributes and its various links to other concepts; an inner formal assertion layer for formally defining the concept [10]. The declaration level is used for conceptual modeling (through a concrete graphical syntax), requirements traceability (through semantic net navigation) and specification reuse (through queries). The assertion level is optional and used for formal reasoning [12].

The generic structure of a KAOS construct is instantiated to specific types of links and assertion languages according to the specific type of the concept being specified. For example, consider the following goal specification for a secret message sending system [9]:

Goal Achieve[RevelationSentToRElay]

Concerns Spy, Revelation, Team, Message

Refines RelayInformed

RefinedTo

RevelationTargetToTeam, TargetedRevelationSentToRelay

InformalDef If a spy collects a revelation, he will send a message about it to his relay

FormalDef $\forall$ sp1, sp2: Spy, re :Team

Collecting (sp1,re) ^Member (sp1,te)^Relay(sp2,te)

$\Rightarrow \Diamond_{\leq 2h}$ ($\exists$ me : Message) Sending (sp1, me, sp2) ^About (me, re)

The declaration part of this specification introduces a concept of type “goal”, named RevelationSentToRelay, stating a target property that should eventually hold (“Achieve” verb), referring to objects such as Spy or Revelation, refining the parent goal RelayInformed, refined into subgoals RevelationTargetedToTeam and TargetedRevelationSentToRelay, and defined by some informal statement.

The optional assertion part in the specification above defines the goal Achieve [RevelationSent] in formal terms using a real-time temporal logic. In this document, the following classical operators for temporal referencing are used [6]:

◦ (in the next state)	• (in the previous state)
◊ (some time in the future)	♦ (some time in the past)
❑ (always in the future)	■(always in the past)
W (always in the future unless)	U (always in the future untill)
B (always in the past back to)	S (always in the past since)

Formal assertions are interpreted over historical sequences of states. Each assertion is in general specified by some sequences and falsified by some other sequences. The notation (H, i) |= P is used to express that assertion P is satisfied by history H at time position i (i € T), where T denotes a linear temporal structure assumed to be discrete for sake of simplicity. Let use the notation H |= P for (H, 0) |= P.

States are global; the state of the composite system at some time position i is the aggregation of the local states of all its objects at that time position. The state of an individual object instance ob at some time position is defined as a mapping from ob to the set of values of all ob's attributes and links at that time position. In the context of KAOS requirements, a historical sequence of states defines a behavior produced by a scenario [7].

The semantics of the above temporal operators is then defined as follows [3]:

(H, i)|= ◦ P	iff (H, next(i)) |=P
(H, i)|= ◊ P	iff (H,j) |=P for some j ≥i
(H, i)|= ❑P	iff (H,j) |=P for all j ≥i
(H, i)|= PUQ	iff there exists a j≠I such that (H, i)|=Q and for every k, i≤k<j, (H,k) |=P
(H, i)|= PWQ	(H,i) |=PUQ or (H,i) |=❑P
(H, i)|= PSQ	iff there exists a j≤I such that (H, i)|=Q and for every k, i<k≤j, (H,k) |=P
(H, i)|= PBQ	(H,i) |= PSQ or (H,i) |=■P

Note that ❑P amounts to PW false. Let use the standard logical connectives $\wedge$ (and), $\vee$(or), $\neg$ (not), $\rightarrow$(implies), $\leftrightarrow$ (equivalent), $\Rightarrow$ (strongly implies), $\Leftrightarrow$ (strongly equivalent), with

$$P \Rightarrow Q \text{ iff } \Box (P \rightarrow Q)$$
$$P \Leftrightarrow Q \text{ iff } \Box (P \leftrightarrow Q)$$

Note thus that there is an implicit outer $\Box$- operator in every strong implication. Beside the agent-related classification of goals, goals in KAOS are also classified according to the pattern of temporal behavior they capture:

Achieve: $C \Rightarrow \Diamond T$
Cease: $C \Rightarrow \Diamond \neg T$
Maintain: $C \Rightarrow TWN$, $C \Rightarrow TWN$
Avoid: $C \Rightarrow \neg TWN$, $C \Rightarrow \neg T$

In these patterns, C, T, and N denote some current, target, and new condition respectively. In requirements engineering, it is needed to introduce real-time restrictions. Bounded versions of the above temporal operators are therefore introduced in the following style:

$\Diamond \leq d$ (some time in the future within deadline d)
$\Box \leq d$ (always in the future up to deadline d)

To define such operators, the temporal structure T is enriched with a metric domain D and a temporal distance function dist: T x T $\rightarrow$ D, which has all desired properties of metrics. We will take:

T: the set of naturals
D: {d | there exists a natural n such that d=n×u}, where u denotes some chosen tin}
dist(i,j): $|j-i| \times u$

Multiple units can be used – e.g., s (second), m (minute), d (day), etc; these are implicitly converted into some smaller unit. The o-operator then yields the nearest subsequent time position according to this smallest unit.

The semantics of the real-time operators is then defined accordingly, e.g.,

$(H, i) \models \Diamond_{\leq d} P$ iff $(H,j) \models P$ for some $j \neq i$ with $dist(i,j) \leq d$

$(H, i) \models \Box_{\leq d} P$ iff $(H,j) \models P$ for all $j \neq i$ such that $dist(i,j) < d$

In the above goal declaration of RevelationSentToRelay, the conjunction of the assertions formalizing the subgoals RevelationTargetedToTeam and TargetedRevelationSentToRelay must entail the formal assertion of the parent goal RevelationSent they define together. Every formal goal refinement thus generates a corresponding proof obligation [13].

In the formal assertion of the goal RevelationSentToRelay, the predicate Sending(sp1, me, sp2) means that, in the current state, an instance of the Sending relationship links variables sp1, me, and sp2 of sort Spy, Revelation, respectively. The Sending relationship and Revelation entity are defined in such a way:

Entity Revalation
InformalDef Secret information about the enemy
Has Content: Text

Relationship Sending
InformalDef A spy is sending a message to another spy
Links Spy {Role Sends}
Message {Role Sent}
Spy{Role To}
The Spy type might in turn be declared by
Agent Spy
InformalDef Person who is an employee of the a spy agency
Has Name: Text

In the declarations above, Name is declared as an attribute of the entity Spy.

As mentioned earlier, operations may be specified formally by pre and post conditions in the state-based style, e.g.,

Operation Send Relevation
Input Spy {arg sp1, sp2}, Revelation {arg re}
Output Message {res me}, Sending
DomPre (me: Message) Sending (sp1,me,sp2)
DomPost (me: Message) Sending (sp1,me,sp)

The pre and post condition of the operation SendRevelation above are domain properties; they capture corresponding elementary state transitions in the domain, namely, from a state where no message is sent to a state where a message is sent. The software requirements are found in the terminal goals assigned to agents in the software-to-be, and in the additional pre, post and trigger conditions that need to strengthen the corresponding domain conditions in order to ensure all such goals. Assuming the RevelationSentToRelay goal is assigned to the spy collecting the revelation one would derive from the above formal assertion for that goal:

Operation SendRevelation
...
ReqTrigFor RevelationSent
Collecting (sp1, re)
ReqPreFor RevelationSent
(∃ te: Team) Member (sp1, te) ^Relay (sp2,te)
ReqPostFor RevelationSent
About (me,re)

The trigger condition captures an obligation to trigger the operation as soon as the condition gets true and provided the domain precondition is true. The specification will be consistent provided the trigger condition and required precondition are together true in the operation's initial state.

The Elaboration Method

Figure 6.1 outlines the major steps that may be followed to elaborate KAOS specifications from high-level goals.

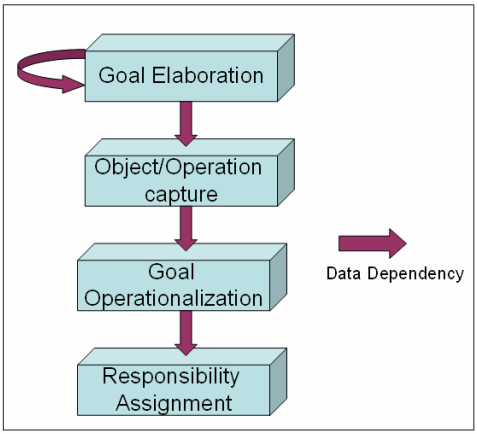


Figure 6.1. KAOS requirements elaboration

Goal elaboration. Elaborate the goal AND/OR structure by defining goals and their refinement links until assignable goals are reached. The process of identifying goals, defining them precisely, and relating them through refinement links is in general a combination of top-down and bottom-up subprocesses; offspring goals are identified by asking HOW questions about goals already identified whereas parent goals are identified by asking WHY questions about goals and operational requirements already identified [12].

Object capture. Identify the objects involved in goal formulations, define their conceptual links, and describe their domain properties by invariants [13].

Operation capture. Identify object state transitions that are meaningful to the goals. Goal formulations refer to desired or forbidden states that are reachable through state transitions; the latter correspond to applications of operations. The principle is to specify such state transitions as domain pre- and post conditions of operations thereby identified and to identify the agents that could perform these operations [3].

Operationalization. Derive strengthened pre-, post-, and trigger conditions on operations and strengthened invariants on objects, in order to ensure that all terminal goals are met. A number of formal derivation rules are available to support the operationalization process [2].

Responsibility assignment. 1) Identify alternative responsibilities for terminal goals; 2) make decisions among refinement, operationalization, and responsibility alternatives, so as to reinforce nonfunctional goals e.g., goals related to reliability, performance, cost reduction, load reduction, etc; 3) assign the operations to agents that can commit to guarantee the terminal goals in the alternatives selected. The boundary between the system and its environment is obtained as a result of this process and the various terminal goals become requirements or assumptions dependent on the assignment made [38].

The steps above are ordered by data dependencies; they may be running concurrently, with possible backtracking at every step.

Obstacle Analysis and Resolution

Obstacles were first introduced in [2] as means for identifying goal violation scenarios. First-sketch specifications of goals, requirements, and assumptions tend to be too ideal; they are likely to be occasionally violated in the running system due to unexpected agent behavior. The objective of obstacle analysis is to anticipate exceptional behaviors in order to derive more complete and realistic goals, requirements, and assumptions. A defensive extension of the goal-oriented process model outlined above is depicted in Figure 6.2. During elaboration of the goal graph by elicitation and by refinement, obstacles are generated from goal specifications. Such obstacles may be recursively refined as indicated by the right circle arrow in Figure 6.2.

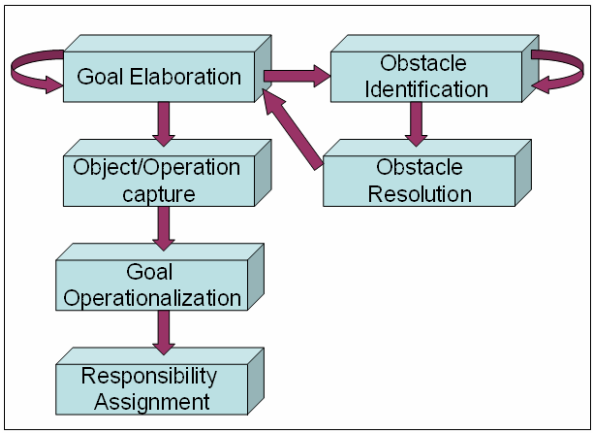


Figure 6.2. Obstacle analysis in goal-oriented requirements elaboration

In declarative terms, an obstacle to some goal is a condition whose satisfaction may prevent the goal from being achieved. An obstacle O is said to obstruct a goal G in some domain characterized by a set of domain properties Dom if and only if

$$\{O, Dom\} \models \neg G \qquad \text{obstruction}$$
$$Dom \not\models \neg O \qquad \text{domain consistency}$$

Obstacle analysis consists in taking a pessimistic view at the goals, requirements and expectations being elaborated. The principle is to identify as many ways of obstructing them as possible in order to resolve each such obstruction when likely and critical so as to produce more complete requirements for more robust systems. Formal techniques for generation ad AND/OR refinement of obstacles are detailed in [2, 13].

The basic technique amounts to a precondition calculus that regresses goal negations $\neg G$ backwards through known domain properties Dom. Formal obstruction patterns may be used as a cheaper alternative to shortcut formal derivations. Both techniques allow domain properties involved in obstructions to be incrementally elicited as well [3].

Obstacles that appear to be likely and critical need to be resolved once they have been generated. Resolution tactics are available for generating alternative solutions, notably, goal substitution, agent substitution, goal weakening, goal restoration, obstacle prevention and obstacle mitigation [2]. The selection of preferred alternatives depends on the degree of criticality of the obstacle, its likelihood of occurrence and on high-priority soft goals that may drive the selection. The selected resolution may then be deployed at specification time, resulting in specification transformation, or at runtime through obstacle monitoring. Obstacle resolution results in a goal structure updated with new goals and/or transformed versions of existing ones. The new goal specifications obtained by resolution may in turn trigger a new iteration of goal elaboration and obstacle analysis. Goals obtained from obstacle res`olution may also refer to new objects/operations and require specific operationalizations [8].

6.2 Formal Analysis and Design for Security Engineering

Background

Security is a growing concern as the software community increasingly develops larger and more complex systems. These systems support ever more distributed and integrated capabilities in the public and private sectors. As society increasingly depends on software; the size and complexity of software systems continues to dynamically grow and evolve with ever-richer semantics making them more difficult to structure and understand. Trusted system requirements compound the problem by adding security and privacy dimensions to the mix, and most software efforts become untenable as software organizations attempt to bolt on security mechanisms. One of the major sources of security vulnerabilities has been poor- quality software [18]. Security aspects are usually applied to products late in the development cycle leaving systems vulnerable to attacks. This not only results in ineffective security capabilities as seen in the relentless barrage of cyber attacks, but also the integrity of software systems is placed at risk

as the engineering principles used to develop software systems are subverted by the subsequent burden of security requirements [4].

After three decades of adding security capabilities to software systems, two patterns arise. First is a heavy reliance on the latest security gadgets and post-development security evaluations [Common Criteria (CC) Security Evaluations] to provide confidence in trusted systems. The second is an unbalanced dependence on legal remedy for responding to security responsibility claims - citing "the corporation provided security measures that are inline with current practice". This will not be much good when a national disaster is caused by system vulnerabilities that could have been averted by engineering for security.

Engineering security requirements is a message that has recently received more attention from the research community [4] due to the losses caused by poorly secured software products that result from considering security as an afterthought of software development. According to Van Lamsweerde, there are three reasons for considering security requirements after the fact. First, early phases of software development raise the priority of analyzing and elaborating functional requirements over non-functional requirements such as security in order to obtain a functioning product in a short amount of time. The second reason is the lack of a constructive and effective mechanism for elaborating security requirements in a complete, consistent, and clear manner. The third reason is the lack of a precise and well-defined approach to produce design and implementation of security requirements that accurately achieve these requirements while ensuring proper handling of all requirements and allowing for requirements traceability at the different phases of development. The Formal Analysis and Design for Security Engineering (FADSE) approach [1] to engineer security requirements addresses these problems that hinder the consideration of security requirements at the various stages of software development.

In Figure 6.3, Wing depicts a layered approach to build secure systems [19]. The cryptographic layer provides primitives for encryption, decryption, digital signatures, etc. The protocols layer

provides means for securing communication including authentication and exchange protocols. The systems and languages layer provides security services built in general-purpose programming languages such as C or Java. The applications layer include applications like online shopping, banking, etc. to guarantee some level of privacy and protection to users' data [19].

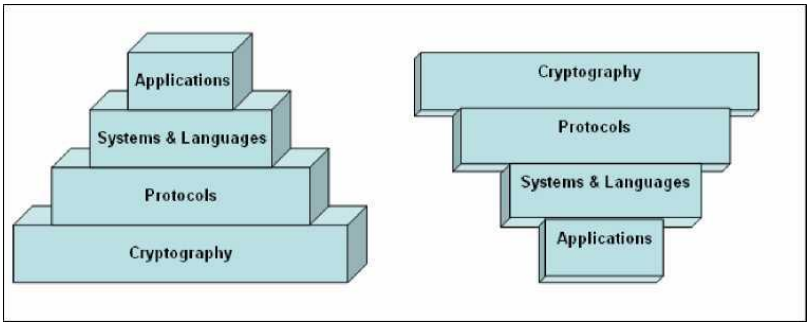


Figure 6.3 A layered approach to build secure system by Wing
a: System Layers; 3b: Security Guarantees

Figure 6.3b illustrates an ironic observation of the inverse proportionality between the strength of what we can guarantee at each layer in the security-layered approach to the size of the layer (Figure 6.3b). Wing suggests that there are significant results in cryptography, which can accurately tell us what we can guarantee and what we cannot. At the protocol level, formal methods have proven successful in providing guarantees for authentication protocols while at the systems and languages layer, commercial technology like Active X and Java provide different levels of security, but are still open to public attacks such as denial of service and spoofing. However, at the application layer in terms of security guarantees, there is not much to provide a reasonable level of security [19].

FADSE described excels in applications layer to fulfill the essential need for a formal security requirements elaboration method encapsulated in an uncomplicated interface and transformed to a formal

language to rigorously derive design specifications that maintain properties of the requirements model.

Security requirements engineering entails developing methods and tools which support the construction of complete, consistent, and clear specifications describing what a software system under development is supposed to do [20]. Candidate solutions including semi-formal modeling notations centered on object-oriented analysis (OOA) techniques like UML and formal specification techniques have been proposed to address the requirements problem.

The graphical notations provided with the semi-formal approaches are easy to use and communicate [3]. Further, the different kinds of diagrams may provide complementary views of the same system; such views can be related to each other through inter-view consistency rules [21]. However, semi-formal approaches to security engineering have limitations in that they can only cope with functional aspects; they generally support highly limited forms of specification and analysis; the semi-formal notation is most often fairly fuzzy since the same model may often be interpreted in different ways by different people [3].

Formal methods are complete and precisely defined, but need mathematical skills for effective use. Therefore most security specification tasks are still carried out with the support of informal specification methods, though this practice may lead to dangerously ambiguous, inconsistent, or incomplete specifications resulting in poorly secure software systems. Despite their advantages in solving security problems, formal methods are in fact still perceived as too cumbersome and complicated to be generally applied, and are relegated to the most critical sections of software development and software systems. Being a signification part of the critical aspects in software, security concerns are suitable candidates for development using formal methods. Nevertheless, a good chance for wider acceptance and use is given, if sufficient approach and tool support encapsulating and hiding the formal part become available.

Employing formal methods in real systems is steadily growing from year to year [22]. Despite of the good news, traditional semi-formal

(unlike the goal-oriented Knowledge Acquisition for automated Specifications (KAOS) approach) modeling and formal specification techniques suffer from serious shortcomings that explain why they are not fully adequate for the critical phase of requirements elaboration and analysis. These shortcomings are outlined as follows [4]:

- *limited scope:* The vast majority of techniques focus on the modeling and specification of the software alone. They lack support for reasoning about the composite system made of the software and its environment;
- *lack of rationale capture:* Formal notations do not address the problem of understanding requirements in terms of their rationale with respect to some higher-level concerns in the application domain;
- *poor guidance:* Constructive methods for building correct models/specifications for complex systems in a systematic, incremental way are by large non-existent. The problem is not merely one of translating natural language statements into some semi-formal model and/or formal specification. Requirements engineering in general requires complex requirements to be elicited, elaborated, structured, interrelated and negotiated;
- *lack of support for exploration of alternatives:* Requirements engineering is much concerned with the exploration of alternative system proposals. Different assignment of responsibilities among software/environment components yields different software-environment boundaries and interactions. Traditional modeling and specification techniques do not allow such alternatives to be represented, explored, and compared for selection.

The above limitations have been overcome in the FADSE approach through the employment of goal-orientation during requirements analysis [4, 2, 23]. There are a number of goal-oriented approaches to requirements engineering. Knowledge Acquisition for automated Specifications (KAOS) and i* represent the state-of-the-art as mature approaches in the goal-oriented requirements engineering paradigm

[24]. KAOS, the more well- established of the two, uses first-order temporal logic as its formal infrastructure with good tool support [23]. Further, KAOS is unique in its conceptual ontology: lower level descriptions of the system-to-be are progressively derived from system-level and organizational objectives using a framework that is essentially a taxonomy of concepts instantiated for a particular domain [25].

The KAOS security extension employed in FADSE to model and analyze security requirements meet a set of meta-requirements that make such model securely reliable as follows [4]:

- *early deployment:* In view of the criticality of security requirements, the technique is applicable as early as possible in the requirement engineering process, that is, to declarative assertions as they arise from stakeholder interviews and documents (as opposed to, e.g., later state machine models);
- *incrementality:* This technique supports the intertwining of model building and analysis and therefore allow for reasoning about partial models;
- *reasoning about alternatives:* This technique makes it possible to represent and assess alternative options so that a "best" route to security can be selected;
- h*igh assurance:* This technique allows for formal analysis when and where needed so that compelling evidence of security assurance can be provided;
- s*ecurity-by-construction:* To avoid the endless cycle of defect fixes generating new defects, the requirements engineering process is guided so that a satisfactory level of security is guaranteed by construction;
- s*eparation of concerns*: This technique keeps security requirements separate from other types of requirements so as to allow for interaction analysis.

FADSE is the first to extend the goal-oriented KAOS framework to formal design and implementation, which brings the benefits of the goal-oriented paradigm to the software security domain. The employment of goal-orientation prior to stepping into formal design

paves the road for formal design through performing thorough reasoning about security requirements and organizing them into a well-structured requirements model. Van Lamsweerde argues that goals offer the right kind of abstraction to address the inadequacies of formal and semi-formal methods for requirements engineering (especially for high assurance systems). These systems require compelling evidence that they deliver their services in a manner that satisfies safety, security, fault-tolerance and survivability requirements.

Goal-oriented methods are adequate for requirements engineering that is concerned with the elicitation of goals to be achieved by the software-to-be (WHY issues), the operationalization of such goals into specifications of services and constraints (WHAT issues), and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software available or to be developed (WHO issues) [26]. Positive/negative interactions with the other system goals can be captured in goal models and managed appropriately [2]; exceptional conditions in the environment that might hinder critical goals from being achieved can be identified and resolved to produce more robust requirements [2]; the goals can be specified precisely and refined incrementally into operational software specifications that provably assure the higher-level goals [8, 7, 11]. Requirements implement goals much the same way as programs implement design specifications [4].

Most research efforts in the field of security requirements engineering have been devoted to the requirements specification facet of requirements engineering [14]. A large number of languages have been proposed for requirements specifications, some of which are popular formal languages like Z, VDM, or LARCH. However, these languages are not well suited for capturing requirements models because they are too restricted in scope; they address only the “what” questions [11]. Typically, the data and operations of the system envisioned are specified through first-order assertions like conditional equations or pre/postconditions and invariants. Another limitation is that such languages have no built-in constructs for making a clear

separation between domain descriptions and actual requirements [14]. Van Lamsweerde indicated that recent attempts to design semi-formal languages like KAOS support a wider range of requirements with the ability to address the “why”, “who”, and “when” questions in addition to the normal “what” questions [14].

FADSE addresses the limitations of the security requirements specification languages through employing KAOS to elicit security requirements that uses first order temporal logic to formally capture pre, post conditions and domain invariants. Moreover, KAOS models security aspects as goals resulting in a security goal graph in which the bottom level goals are either security requirements assigned to agents in the software-to-be (the software whose requirements are being modeled) or assumptions to be fulfilled by the interacting environment. This differentiation between requirements and assumptions clearly separates domain descriptions from actual requirements. Further, the bottom level goals of the graph are goals to be assigned to agents allowing the requirements model to extend beyond answering what questions to answer how questions and who is responsible for achieving these goals.

Being a crucial aspect in system development, security must be thoroughly ensured during all development phases. While still mostly relegated to the implementation and testing phases, it should be enforced at earlier stages too, i.e. in the requirements elaboration phase since early detection of possible security vulnerabilities is a key factor towards developing secure systems that are cost effective. Again, FADSE is a requirements-driven approach that considers security very early in development while security vulnerabilities are analyzed and resolved. Moreover, the rest of the development phases are guided requirements model using the B formal method refinement mechanism that obtains its initial B model from automatically transforming the KAOS model to B. The employment of B as a design elaboration method to fully develop security-specific elements of software allows for the automatic verification of security implementation from more abstract properties represented in the requirements model. Further, the

derivation of acceptance test cases from the requirements model provides means to ensure compliance between the derived implementation and the initial set of security requirements.

Automatic code generation from requirements facilitates the production of large systems with high- quality in a cost-effective manner. Therefore, it has been one of the objectives of software engineering almost since the advent of high-level programming languages, and calls for a "requirements-based programming" capability has become deafening [27]. Several tools and products exist in the marketplace for automatic code generation from a given model. However, they typically generate code, portions of which are never executed, or portions of which cannot be justified from either the requirements or the model [27]. Moreover, existing tools do not and cannot overcome the fundamental inadequacy of all currently available automated development approaches, which is that they include no means to establish a provable equivalence between the requirements stated at the outset and either the model or the code they generate. FADSE solves this problem by building a mature and formally analyzed security requirements model using KAOS and transforms this requirements model into B to fully develop security aspects. Discharging B proof obligations provides means to establish a provable equivalence between the security requirements model and the more specified models produced for design and implementation.

While security-in-the-large encompasses development, deployment, and administration of trusted systems in their operational environments, this research focuses largely on the development process. We conjecture that *purposefully engineering security principles into trusted systems during development reduces the risks of manifesting vulnerabilities during deployment, administration, and operation in the trusted environment*. More concisely, while well understood security principles exist to guide organizations and security best practices exist for specific areas of development, there is no cohesive framework of security engineering principles that integrates development activities, artifacts, and practices, with relevant security principles. Lacking an

integrated software security engineering process that incorporates security principles from the onset, we continue to employ informal solutions that render many subsequent systems vulnerable to attacks.

From the perspective of building a system devoid of security vulnerabilities, formal methods have been hoisted up as a reasonable solution. However, formal methods come with some challenges for large, complex systems. First, the application of formal methods entails substantial requirements formulation in a precise provable and correct representation, unambiguous, and consistent. Even for moderate systems, this goal is not readily achieved from a cost and time perspective. Secondly, once formal requirements set exists, translating them into a design that preserves the requisite security properties can be arduous and error-prone and arduous. Thirdly, the availability of engineers with the requisite experience to render a system definition and design specification formally is relatively low and they are often more expensive to employ.

Combining the increasing need for secure systems and the challenges of formal methods-based solutions, we have a dilemma. Do we bite the bullet and apply formal methods to engineering secure systems or do we continue down the informal path and live with the resulting vulnerabilities? This approach leads us to believe that there is a rational middle ground. First, assuring that all vulnerabilities are covered is unrealistic both technically and economically unless we are willing to accept systems of low size and complexity. Secondly, requirements engineering with formal methods is precise and provides a good platform for better completeness, but it still lacks the guarantees that completeness is achieved. Therefore, the argument for completeness turns to a tradeoff between approaches for time taken for getting to an acceptable level of completeness before proceeding on to design.

The requirements-driven goal-directed FADSE approach proposed as a step towards the development of highly secure software is less precise and formal than starting from a totally formal approach, but it has three key mechanisms that render near-formal results. First, it

employs KAOS (Knowledge Acquisition for autOmated Specifications) [4, 2], which is a proven goal-directed framework to elaborate security requirements. KAOS is natural for identifying and reasoning about the requisite requirements - making it an effective mechanism for facilitating completeness with additional formality in areas where it is essential. Second, the obstacle analysis mechanism of KAOS provides a good capability to reduce the presence of vulnerabilities by concentrating on the most appropriate alternatives for avoiding or eliminating the obstacles. Third, the test-case generation from the requirements model assures better alignment with the requirements and confidence in the specification from a verifiability perspective.

FADSE might have a broader impact in two dimensions. The first is that it could enhance the infrastructure for research as some of the case studies used to demonstrate and verify the approach are industry-oriented. This collaboration between the proposed work and industry broadens its impact and extends its applicability and practicality. The second dimension in which FADSE has a broader impact is that it widens dissemination to enhance scientific and technological understanding. This is achieved through the results obtained from verifying the approach through a controlled experiment, which are beneficial to software engineers in general and to security engineers in specific who lack a constructive and systematic approach to capture and develop security concerns in software products.

State of the Art Survey

FADSE is a security engineering approach employing formal methods to analyze, design and implement security-specific elements of software with the aim of producing highly secured software products. FADSE can therefore be positioned in the intersection of three areas of software research namely software engineering, security and formal methods as illustrated in Figure 6.4. This chapter surveys research efforts of employing formal methods to produce highly secured software, elaborating and modeling security requirements, deriving design specifications and implementation from requirements using

formal and semi-formal methods and checking consistency between requirements and design.

Clarke, et. al. defined formal methods as “mathematically based languages, techniques and tools for specifying and verifying software systems” [28]. They have argued that using formal methods does not necessarily guarantee correctness, but they can significantly elevate our understanding of the system through detection of inconsistencies, ambiguities and incompleteness. Using formal methods for producing highly assured software has long been accepted and advised for secure systems [27]. Even contemporary security evaluation suggests formal specification and controlled transformation as evidenced in the Common Criteria Evaluation Assessment Levels (EAL) 5, 6 and 7 requiring formalism. Yet, formal methods are rarely exercised in the domain of security engineering.

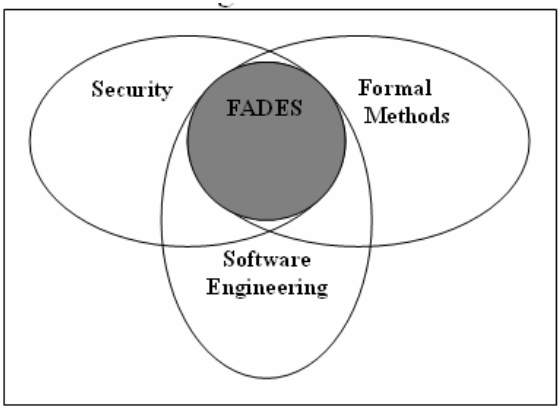


Figure 6.4. FADSE Position in Software Research

The Common Criteria (CC) is an international standard for evaluation of software security. It provides a framework for security users to specify their security requirements while software vendors implement and make claims about the security attributes of their products that are evaluated by the concerned parties like test

laboratories to verify the claims. The CC defines seven Evaluation Assurance Levels (EAL 1 to EAL 7) to measure the degree of compliance between the product and the claimed security functionality. Each EAL covers the complete development of a product with a given level of strictness with EAL 1 being the most basic and hence the cheapest to implement and evaluate and EAL 7 being the most rigorous and expensive [29]. The first four EAL levels of the CC do not require formal evidence for assuring the security functionality of the products while EAL levels 5, 6 and 7 have requisites of providing formal artifacts of development to assure the claimed security attributes. This means that products assured at EAL 5-7 provides more confidence to the users in the security claims of the product.

EAL1 is applicable where some confidence in correct operation is mandated while security threats are not considered serious [29]. Evaluation at EAL1 should show evidence of compliance between the target of evaluation functions and its documentation using, for example, testing. EAL2 requires the interference of the developers in delivering design and test results and is applicable where low to moderate level of security is required. EAL2 provides security assurance by analyzing the security functions using requirements specifications, guidance documentation and the high-level design of the target of evaluation [29]. EAL3 thoroughly investigate the target of evaluation without substantial re-engineering. EAL3 provides security assurance using requirements specifications, documentation and the high level design; however, the analysis is supported by testing of security functions and evidence of the developer testing. EAL4 is applicable to moderate and high level security and it provides assurance through analysis of security functions using complete requirements specifications, documentation, high-level and low-level design, and subset of the implementation to understand security behavior [29].

EAL5 is the first assurance level requiring formal evidence of development to assure high-level security of the target of evaluation. EAL5 requires complete specification, documentation, high-level and low-level designs and all of the implementation. Assurance is further

gained through a formal model of the security policy and a semi-formal presentation of the requirements specifications and the high-level design and a semi-formal evidence of the compliance between the two. EAL6 gains assurance through a formal model of security, a semiformal presentation of the security requirements specifications, high-level and low-level designs and a semiformal evidence of the compliance between the specifications and the two designs. Further, a modular and layered design is required. EAL7 is applicable to extremely high-risk applications where the assets need maximum protection. EAL7 assures security through a formal model of the security policy, formal presentation of the security requirements specifications, high-level design, a semiformal presentation of the low-level design and formal and semiformal evidence of compliance between specifications and both the high-level and low-level designs. These definitions imply that the software products developed with FADSE could be assured at EAL5, EAL6 and EAL7. These three evaluation assurance levels especially EAL6 require semiformal presentation of the requirements specifications, which are provided in FADSE in the semiformal form of the KAOS goal graph. Further, EAL6 and EAL7 require formal presentation of design, which is provided in FADSE using the B formal method and a semiformal evidence of compliance between specifications and design, which is provided in FADSE using the acceptance test cases generated from the specifications model (KAOS goal graph). The employment of formal methods in FADSE provides the required formal evidence of development for the assurance of the resulting target of evaluation at EAL5-7.

There are a number of success stories for applying formal methods to security-critical systems at the application layer; however, there are few if any formal methods-based approaches that handle security-specific elements. Stepney used the Z formal language to construct a security requirements specifications model that has been further refined to derive design and implementation specifications for a money exchange system using smart cards [30]. Clarke surveyed several

applications of formal methods to security aspects [28]. Oxford University and IBM Hursley Laboratories collaborated in the 1980s on employing Z to formalize part of IBM's Customer Information Control System (CICS), an online transaction processing system with thousands of installations worldwide [28]. IBM reported measurements collected throughout development to indicate an overall improvement in product quality, a reduction in the number of errors discovered, earlier detection of errors, and an estimated 9% reduction in total development cost. Sabatier and Lartigue reported on industrial smart card application in which they designed the transaction mechanism to provide secure means for modifying data that is permanently stored in smart cards [31]. They demonstrated how the use of the B method increased confidence and provided mathematical proof that the design of the transaction mechanism fulfills the security requirements.

A number of approaches have been proposed in the literature for elaborating and modeling security requirements in a way comparable FADSE. Misuse cases that complement UML are able to capture attacker features at requirements engineering time. Misuse cases are defined as "the inverse of UML use cases specifying functions that the system should not allow" [32, 33]. Misuse cases refer to scenarios that result in loss for the organization or some specific stakeholder. The concept of mis-actor is associated with the misuse cases. A mis-actor is defined as "the inverse of a UML actor, who is someone-intentionally or accidentally initiates misuse cases and whom the system should not support in doing so" [32]. Misuse cases can be normally related to normal use cases through "includes", "extends", "prevents", and "detects" relations. The relation of "includes" or "extends" from a misuse case to an ordinary use case indicates a misuse of one of the functions of the ordinary use cases. For example, a denial of service (DoS) attack needs not include illegal actions, just flooding the system with a heavy burden of publicly available registration requests. The "prevents" and "detects" have been introduced specifically for misuse cases, which relate ordinary uses cases to misuse cases outlining functions that prevent or detect misuse [32].

Firesmith in [34] has differentiated between misuse cases and security use cases. Firesmith thinks that misuse cases provide means for analyzing security threats but are inappropriate for the analysis and specification of security requirements. Instead, security use cases should define possible threat scenarios from which the application should be protected. Figure 6.5 shows the functionality of both misuse cases and security use cases. Security use cases could then be integrated with the rest of UML artifacts such as class and interaction diagrams in order to provide a software engineering approach that captures security requirements early on and integrates them with the rest of the system development phases through UML [34].

Unlike FADSE, misuse cases and security requirements use cases handle security concerns only during the requirements and analysis phases while FADSE covers the different stages of development for security requirements to produce a complete implementation. Misuse cases and security use cases inherit the simplicity and popularity as well as the semantic inconsistencies of UML. Further, they lack the rigor and requirements traceability features in FADSE.

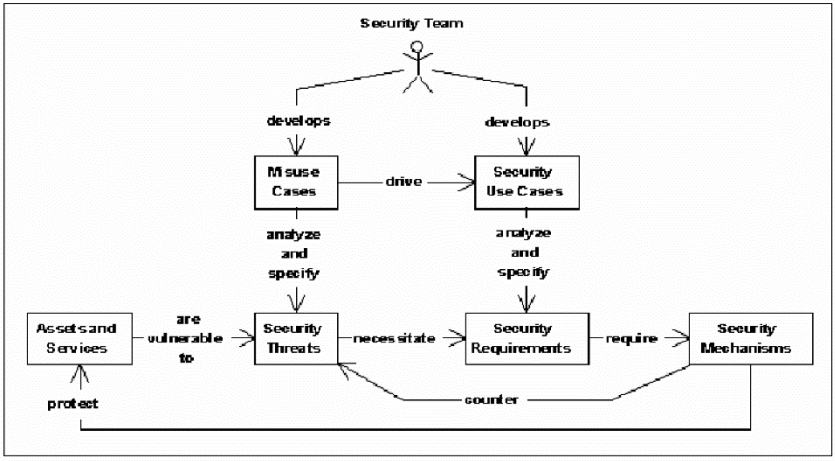


Figure 6.5. Misuse cases vs. security use cases [17]

Liu et. al. have extended the i* agent-oriented requirements modeling language to handle security and privacy concerns [35, 36]. The i* is an agent-oriented framework for modeling and redesigning intentional relationships among actors that are strategic to the software being modeled [37]. The i* framework is concerned with the early phase of requirements engineering with the notion of strategic actor being a central concept. Actors have properties like goals, beliefs, abilities, and commitments. The framework focuses on analyzing the strategic implications that each actor is concerned with in order to meet that actors' interests. This is achieved through modeling intentional relationships among actors, rather than input/output data flow, in which actors depend on each other to achieve goals, perform tasks, or employ resources. Modeling software from the agent perspectives has shown potential in extending the i* framework to model security aspects that originate from human concerns and intents; therefore, should be modeled through social concepts [35].

This i* security extension provides analysis techniques to deal with security requirements. The first technique is the attacker analysis that

identifies potential system attacks. The attacker analysis theme is that "all actors are assumed guilty until proven innocent" [35]. Ordinary system actors (roles, positions or agents) are considered among potential attackers to the system or to other actors. The second analysis technique is the dependency vulnerability analysis that detects vulnerabilities in terms of organizational relationships among stakeholders whose dependency relationships bring vulnerabilities to actually attack the system through the manipulation of their malicious intents. Detailed analysis of vulnerability with the i* dependency modeling capability to trace the potential failure of each dependency to a dependent and to its dependents. Countermeasure analysis proposes proactive actions to resolve vulnerabilities and threats. Finally, access control analysis fills in the gap between security requirements and their realization in implementation. Access control analysis uses i* models to embed a proposed solution to system design. The i* role-based requirements analysis with i* facilitates the transition from requirements to design since it fits naturally to the role-based access control methodology software design.

The i* security extension is close in spirit to KAOS security extension employed in FADSE. The main difference is that the i* security extension starts with agents involved in the system rather than goals being threatened as in KAOS. Further, the i* security extension identifies insider attackers only, that is, system stakeholders that were identified before in the primal model and might be suspect while KAOS identifies possible attacks regardless they can be performed by insider or outsider attackers. In the i* security extension, the malicious goals owned by attackers are not modeled explicitly and the methodology provides no formal techniques for building threat models. On the other hand, the KAOS security extension provides a formal procedure for generating attacks and countermeasure to such attacks, which makes KAOS more reliable especially in the security requirements context.

Both the KAOS security extension and the i* security extension provide constructs to effectively reason about security requirements. However, the KAOS security extension has been favored over the i*

one to employ in FADSE because the i* framework is not based on first-order predicate logic like KAOS making it difficult to transform to a formal language like B.

There are a number of approaches proposed in the domain of deriving design from requirements both formally and semi-formally. Some of these approaches extended the KAOS framework for further stages in development like architecture and design. All the approaches that extended KAOS exploited the key features of goal-orientation namely the ability to explore alternatives in specifications, responsibility assignment, goal formalization, adapting different levels of formality, and modeling the software and its environment. However, the approaches that extended KAOS focused on the functional and non-functional requirements of the system and none of them addressed the security aspects like FADSE. This distinguishes FADSE in being the first to employ goal-orientation for the design and implementation of security-specific aspects of software system.

Nekagawa et. al. have proposed a formal specification generator that transforms KAOS requirements specifications into VDM++ specifications to develop software systems [38]. Requirements are elaborated and analyzed using KAOS resulting in the construction of a requirements model, which is given as an input to the generator to produce the VDM++ specification. Missing parts in the requirements model are commented during the generation process to prompt developers to augment the VDM++ specification. The generated specification contains implicit operations consisting of pre and post-conditions, inputs, and outputs while the body of operations is left for developers to add to create an explicit specification. Test cases are developed to verify the formal specification using the VDM tools [38]. An overview of the VDM++ generator steps and artifacts is illustrated in Figure 6.6. Like FADSE, the VDM++ generator formally derives design from requirements modeled with KAOS. Nevertheless, the VDM++ generator only derives high level design with no specific focus on security aspects.

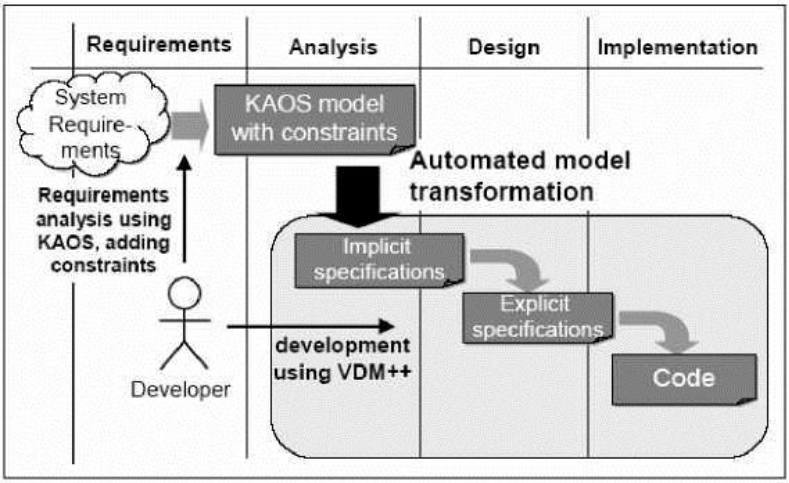


Figure 6.6. Overview of Development Process

Jiang, et. al., presented a case study of a real world industrial application that produced several versions of conceptual schema design for a biological database during its evolution [26]. The case study compares two different methods for designing a database. The case study authors started with an analysis of the original conceptual schema and its evolving design. They then revisited the design process using KAOS in order to construct a goal model of the problem domain. The case study has been used as a proof of concept for the authors' work in devising an extended database design methodology in which stakeholder goals drive the design process in a systematic way. This research direction has been motivated by the authors' belief in goal-oriented capabilities of making stakeholders' goals explicit, and exploring a space of design alternatives that lead to a set of data requirements specifications, each of which corresponds to a particular choice to fulfill the top-level goals

The comparison of the design choices for the biological database as originally made by its designers in successive versions and the design

recommendations suggested by the goal analysis shows that the goal-driven approach results in a design spaces that:

- includes original schema built through the evolution of the application;
- suggests additional alternatives that lead to more comprehensive design;
- supports systematic evaluation of design alternatives;
- generates schemas with rich and explicit data semantics.

The authors' interpretation to the above results is that goal analysis allows the cognitive process that took place in the actual design of the database to be explicit when goals are declared as opposed to being implicit in the traditional method [26]. This led to schema design that better responds to the purpose of the application. The schema resulting from the applying goal analysis provides justification for the design choices and suggests additional alternatives that lead to more comprehensive design in terms of the coverage of the stakeholder goals. Further, an important property of the goal model is that all the alternatives exist to be examined, regardless whether they are selected or not. Goal analysis provides a systematic way for evaluating design alternatives through use of soft goals that are goals without clear-cut criteria for their satisfaction and usually used to model non-functional requirements of a software system [26]. Design alternatives represent different information needs that may have positive or negative contributions to the fulfillment of the soft goals. Moreover, goal-oriented database design provides direct trace from intentions to requirements to schemas. The knowledge captured during design can be used to attach explicit meaning to the elements in the schema and propagate to the data organized by the schema. The justification for the results has been demonstrated with examples from both the traditional schema and the goal-oriented one.

Brandozzi and Perry proposed a method to transform the requirements specification for a software system into an architectural specification [39]. This approach has chosen KOAS as a goal-oriented requirements engineering methodology to specify requirements and

transforms such requirements specifications to APL (Architecture Prescription Language) in order to derive an architecture prescription from the KAOS requirements model. The authors justified their choice of KAOS by their belief that goal- oriented specifications, among all kinds of requirements specifications, are nearer to the way human thinks and are easy to understand by all stakeholders. Another reason is that goal-oriented specifications are particularly suitable to be transformed to formal languages like APL. The construction of requirements in the form of a directed-acyclic graph provides analytical capability to the requirements model and facilitates its transformation to a formal language. The approach tries to find a solution to the transition from requirements to architecture, which has been traditionally one of the most difficult aspects of software engineering [39]. Such transformation is difficult because it transforms the question of what we want the system to do into a basic framework for how to do it.

This approach takes as input goal oriented requirement specifications and returns as output an architecture prescription. An architecture prescription lays out the space for the system structure by restricting the architectural elements (processes, data, and connectors), their relationships (interactions) and constraints that can be used to implement the system [39]. The main advantages of an architecture prescription over a typical architecture description are that it can be expressed in the problem domain language and it is often less complete, and hence less constraining with respect to the remaining design of the system. An architectural prescription concentrates on the most important and critical aspects of the architecture and these constraints are most naturally expressed in terms of the problem space (or business domain, the domain of the problem). An architecture description, on the other hand is a complete description of the elements and how they interface with each other and tends to be defined in terms of the solution space rather than the problem space (or in terms of components such as GUIs, Middleware,

The rules for transforming KAOS constructs to APL constructs are as follows: each object in the requirements generally corresponds to a

component in the architecture. More specifically, and agent object, and active object, corresponds to either a process or a connector. By definition, a process (thread, task) is an active component. What might not be immediately apparent is that also a connector can be an active component. An example of this type of connector is a software firewall. A software firewall is an active entity that checks whether the processes that want to interact satisfy some conditions or not, and allows or denies the interaction among them accordingly.

The events relevant to the architecture of the system are those either internal to the software system or those in the environment that have to be taken into account by the software system. The receiving of a message by a process is an example of internal event. The triggering of an interrupt by a sensor is an example of external event. An event is generally associated to a connector.

An entity, or passive object, corresponds to a data element, which has a state that can be modified by active objects. For example, the speed of a train is a variable (entity) that can be modified by a controller (agent). A relation corresponds to another type of data element that links two or more other objects and that can have additional attributes. An example of relation data is a data structure whose attributes are the type of train, its current speed and its maximum speed (additional attribute). A goal is a constraint on one or more of the components of a software system. Additional components may be derived to satisfy a nonfunctional goal. An example of a constraint deriving from a goal is that a component of the software system of an ATM has to check if the password typed by the user matches the password associated in the system to the ATM card inserted.

The transformation of KAOS to an architecture prescription is closely related to FADSE in that both approaches model requirements using the KAOS framework with FADSE being more focused on security requirements. However, FADSE achieves more in terms of software implementation since requirements are produced in an implementable form while the other approach transforms requirements to architecture only. Further, FADSE verifies that the derived

implementation maintain the security properties specified in the requirements through the correctness-by-construction guarantees associated with the B formal method. In the other approach, requirements are transformed to APL, which is an architectural prescription language with no such rigid formality that guarantees the fulfillment of the requirements specified in the requirements model by the generated architecture.

Van Lamsweerde extended the KAOS framework to systematically derive architectural design from functional and non-functional requirements so that the compliance between architecture and requirements is guaranteed by construction [40]. Software specifications are first derived from the KAOS requirements model followed by deriving an abstract architectural draft from functional specifications. This draft is refined to meet domain-specific architectural constraints. The resulting architecture is then recursively refined to meet the non-functional goals modeled and analyzed during the requirements engineering process [40]. Van Lamsweerde provides means to bridge the gap between requirements and architecture using a rigorous architectural design process that relies on the use of precise descriptions of the software components and their interactions. Although this approach is still in its preliminary stages, Van Lamsweerde has specified some ideal meta-requirements on the derivation process in which the derivation should be:

- systematic so that active guidance could be provided to architects;
- incremental to allow for reasoning on partial models;
- leading to (at best) provable or (at least) arguably “correct” and “good” architectures in order to demonstrate that the derived architecture indeed meets the functional requirements and achieves the non-functional ones;
- highlighting different architectural views like a security view, a fault tolerance view, etc.

The KAOS framework is used to formulate requirements in terms of objects in the real world of the software-to-be, and in a vocabulary

accessible to stakeholders. Required relations between objects are captured in order to model the environment in which these relations are monitored and controlled by the software. The next step after constructing the KAOS requirements model for the software is to derive software specifications that are formulated in terms of objects manipulated by the software, and in a vocabulary accessible to programmers while capturing required relations between input and output software objects. The derivation of software specifications from requirements follows the below rules:

- all goals assigned to software agents are translated into the vocabulary of the software-to-be by introducing software input-output variables;
- relevant elements of the domain object model are mapped to their images in the software's object model;
- accuracy goals modeling non-functional requirements that mandate the mapping to be consistent are introduced, that is, the state of software variables and database elements must accurately reflect the state of the corresponding monitored/controlled objects they represent;
- assign input/output agents to the accuracy goals introduced in step 3- typically, sensors, actuators or other environment agents.

The derived software specifications are assumed to be non-conflicting as conflicts have been managed upstream in the requirements engineering process [2]. The software specifications are used to obtain a first architectural draft from data dependencies among the software agents assigned to functional requirements. These agents become architectural components that are statically linked through dataflow connectors. The procedure for deriving dataflow architecture from the software specifications is as follows:

- for each functional goal, a component is defined to regroup a software agent responsible;
- for achieving the goal with the various operations operationalizing the goal and performed by the agent;

- the agent's interface is defined by the sets of variables the agent monitors and controls;
- for each pair of components C1 and C2, a dataflow connector is derived from C1 o C2 labeled with variable d if and only if d is among C1's controlled variables and C2's monitored variables.

The initial abstract architecture obtained with the above construction rules defines the refinement space in which different alternatives of component refinement exist. The refinement space is first globally constrained by architectural requirements and then different alternatives are explored to refine components and connectors. Van Lamsweerde proposed imposing "suitable" architectural styles in order to refine the dataflow architecture, that is, styles to be documented by applicability conditions such as domain properties and the soft goals they are addressing [40].

Once the abstract dataflow architecture is refined to meet architectural constraints, it gets ready for further refinement to achieve the non-functional goals (quality of service and development goals). Many of these goals impose constraints on component interaction; for example, security goals restrict interactions to limit information flows along the channels; accuracy goals impose interactions to maintain a consistent state between related objects and so forth. The refinement of the architecture to accommodate nonfunctional goals proceeds as follows:

- for each non-functional terminal in the goal refinement graph G, all specific connectors and components that G may constrain are identified;
- if necessary, G is instantiated to those connectors and components;
- for each non-functional goal-constrained connector or component, it is refined to meet the instantiated non-functional goal associated with it. Architectural refinement patterns might be used to drive the refinement process as described in [40].

Van Lamsweerde characterizes his architectural extension of KAOS as systematic and incremental in a mix of qualitative and formal

reasoning to attain software architectures that meet both functional and non-functional requirements. He argues that deriving architecture that is based on goal-oriented requirements analysis allows for making use of the capabilities of the goal-oriented paradigm. The derived architecture is quite able to accommodate non-functional requirements as well as functional requirements, which is not the case in other paradigms like object-oriented analysis and formal methods. Further, the ability to explore the different alternatives for answering "WHAT" questions lies at the core of goal-orientation allowing for constructive guidance to software architects in their design task.

This extension to KAOS to derive architecture from requirements is similar to FADSE in some aspects like employing goal-oriented approaches during requirements analysis in order to derive an architecture using a constructive procedure in the form of transformation rules. However, the generated architecture from Van Lamsweerde's extension to KAOS is not formal like FADSE or APL [26]. Further, FADSE goes further beyond design and produces requirements into an implementable form while Van Lamsweerde's extension of KAOS generates architectural design with no specific focus on security aspects.

Liu and Yu explored integrating the goal-oriented language GRL and Use Case Maps (UCM) in order to derive architectural design from functional and non-functional requirements [41]. The goal-oriented language GRL is used to support requirements modeling using goal and agent-oriented techniques, and to guide the architectural design process. UCM is a scenario-oriented architectural notation used to express the architectural design at each stage of development. UCM support of scenario orientation allows for visualizing the behavioral aspects of the architecture at varying degrees of abstraction and levels of detail.

In the integrated GRL and UCM approach, GRL models are constructed using business goals and nonfunctional requirements that are refined and operationalized, until some concrete design decisions are launched. These design decisions are further elaborated into UCM scenarios that ask "how" questions instead of "what" questions.

Moreover, UCM scenarios describe the behavioral features and architectures of the system in the restricted context of achieving some implicit purpose(s), which basically answers the "what" questions followed by the “why” questions such as [41]:

- "what the system should do as providing an in-coming call service?"
- "what is the process of wireless call transmitting?"
- "why to reside a function entity in this network entity instead of the other?"

The GRL-UCM integrated approach aims to derive an architectural design from requirements through eliciting, refining and operationalizing requirements incrementally until a satisfying architectural design is launched. The general steps of the process are illustrated in Figure 6.7. Unlike FADSE, the integrated GRL and UCM approach only derives high level architectural design without involving any rigor in the derivation. Further, the integrated approach aims at handling system requirements with no specific focus on security requirements.

Mylopoulos et. al. proposed a requirements-driven software development method called TROPOS that supports agent-orientation of software systems [42]. TROPOS adopts the i* modeling framework described in [36], which offers the concepts of actor, goal, and dependency. These concepts are used to model early and late requirements, architectural and detailed design. Tropos main theme is that building a software system that operates in a dynamic environment requires explicit modeling and analysis of this environment in terms of actors, their goals and dependencies on other actors [42]. Tropos supports four phases of software development:

- early requirements, concerned with understanding the problem by studying an organizational setting; the output of this phase is an organizational model which includes relevant external actors, their respective goals and their interdependencies;

- late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities;
- architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies;
- detailed design, where behavior of each architectural component is defined in further detail.

Like FADSE, TROPOS formally derives detailed design specifications from requirements. Unlike FADSE, Tropos does not focus on security aspects but rather on system requirements.

Dromey proposed the GSE method, which formally derives high level architectural design from a set of functional requirements using behavior trees [43]. GSE derives design from requirements through the admission of the prospect that individual functional requirements are regarded as fragments of behavior while a design that satisfies a set of functional requirements is regarded as integrated behavior. Individual functional requirements are formally modeled using behavior trees representation in the GSE approach to enable the transition from requirements to design. Dromey believes that the behavior tree notation solves a fundamental problem of going from a set of functional requirements to a design satisfying those requirements since it provides a clear, simple, constructive and systematic path for this transition [43]. Behavior trees of individual functional requirements may be composed, one at a time, to create an integrated design behavior tree. From this problem domain representation, a direct and systematic transition to a solution domain representation is feasible. The solution domain is represented in the form of component architecture of the system and the behavior designs of the individual components that make up the system. Unlike FADSE, GSE only considers functional requirements, which makes it inappropriate to security requirements that are typically classified as non-functional. Further, GSE derives high level architectural design as opposed to implementation specifications in FADSE.

Hinchey et. al proposed R2D2C (Requirements to Design to Code) approach, which "offers a mechanical transformation of requirements expressed in restricted natural language or in other appropriate graphical notations into a provably equivalent formal model that can be used as the basis for code generation and other transformations" [44]. Requirements might be expressed using scenarios in constrained (domain-specific) natural language, or in a range of other notations (including UML use cases). Requirements are then transformed to a formal model guaranteed to be equivalent to the requirements stated at the outset. The formal model can be expressed using a variety of formal methods and could be subsequently used as a basis for code generation. Currently, R2D2C is using CSP, Hoare's language of Communicating Sequential Processes, which is suitable for various types of analysis and investigation. CSP could be used for full formal implementations and automated test case generation, etc.

R2D2C involves a number of phases, which are reflected in the system architecture described in Figure 6.7. The following describes each of these phases.

- *D1 Scenarios Capture:* Engineers, end users, and others write scenarios describing the functionalities that should be offered by the intended system. The input scenarios may be represented in a constrained natural language using a syntax-directed editor, or may be represented in other textual or graphical forms;
- *D2 Traces Generation:* Traces and sequences of atomic events are derived from the scenarios defined in D1;
- *D3 Model Inference:* An automatic theorem prover is used to infer a formal model, expressed in CSP - in this case, ACL2, using the traces derived in phase 2. Concurrency laws need to be deeply embedded in the theorem prover to provide it with sufficient knowledge of concurrency and of CSP to perform the inference;
- *D4 Analysis:* different types of analysis could be performed based on the formal model making use of the currently available commercial or public domain tools, and specialized tools that are

planned for development. CSP allows for model analysis at different levels of abstraction using a variety of possible implementation environments;

- *D5 Code Generation:* Existing techniques of automatic code generation from formal models are reasonably well understood and could be applied in the R2D2C approach whether using a tool specifically developed for the purpose, or existing tools such as FDR or converting to other notations suitable for code generation (e.g., converting CSP to B) and then using the code generating capabilities of the B Toolkit.

The generated code might be code in a high-level programming language, low-level instructions for (elector-) mechanical device, natural language business procedures and instructions or the like [44].

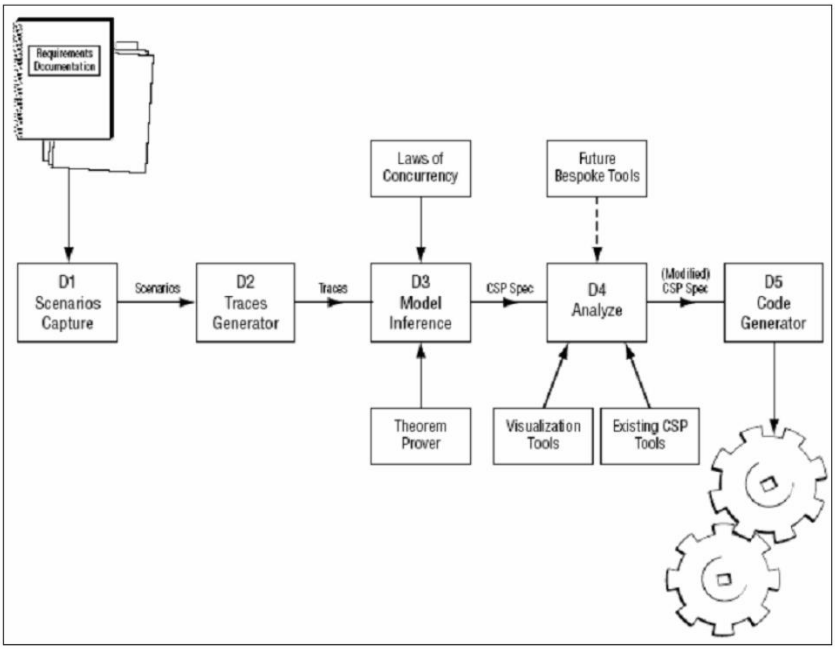


Figure 6.7. The Entire Process with D1 thru D5 Illustrating the Development Approach

R2D2C, the way it is described above, requires significant computing power due to the employment of an automated theorem prover performing significant inferences based on traces input and its knowledge of the concurrency laws. For more applicability of the approach, there is a reduced version of R2D2C called the shortcut version, in which the use of a theorem prover is avoided while maintaining the same level of validity of the approach.

R2D2C has the same spirit as FADSE in terms of building a formal model of requirements from which design and code can be automatically generated. R2D2C employs CSP while FADSE employs KAOS security extension and the B method. Software projects fully developed using B have not reported performance problems that

obstruct the applicability of B to real software due to the level of maturity and stability of the commercial B tools. R2D2C also considers transforming the requirements model specified in CSP to B in order to make use of the code generation capabilities in B tools. Unlike FADSE, R2D2C captures requirements only in scenarios, which cannot capture all types of requirements especially security requirements that might not always be suitable for representation in scenarios.

There are a number of approaches proposed in literature to check for consistency and compliance between requirements and design or implementation. This category of approaches is comparable to FADSE since FADSE allows for the generation of acceptance test cases from the requirements in order to check for the consistency between the derived implementation specifications and the requirements model.

The Ontology for Software Specification and Design (OSSD) approach integrates KAOS and UML to be able to detect errors in software designs against the original requirements [45]. This is achieved through integrating UML with the KAOS framework for elaboration requirements in order to help automate the detection of inconsistencies in UML designs, which enhances the quality of the original design and ultimately integrating the multiple views of UML [45]. OSSD is based on extracting structure, data and relationships from UML design models; abstracts them into an ontology-based integrated model; and creates a specification level representation of the original UML design in a formal, agent-oriented requirements modeling language, which is KAOS.

A simple set of mappings is used to transform the OSSD model to an equivalent KAOS model in order to produce requirements specification that is used as input to an appropriate verification tool in order to detect inconsistencies between the specifications resulted from the OSSD approach and the original requirements. The original UML design is then manually updated based on the results of the verification processing.

“The transformation from UML to OSSD can be summarized as a combined lexical and semantic analysis of the UML Model diagrams,

followed by the utilization of multiple mapping tables that enable the creation of an instance of the OSSD model" [45]. The Upper Merged Ontology (SUMO), WordNet, Browser helps with the categorization of terminology used in the UML diagrams. The first step in the approach is to identify the Object, Attribute, Relation and Behavior Constructs of the OSSD Model using UML class diagrams. Behavior and behavior constraints are refined through the processing of the UML Sequence Diagrams. The processing of the UML State Machine Diagram refines behavior constraints and identifies the States and Transitions in the OSSD Model. Lastly, the OSSD processing of the UML Use Case Diagram identifies the Goals associated with Objects and Behavior in the OSSD Model. The OSSD is depicted in Figure 6.8.

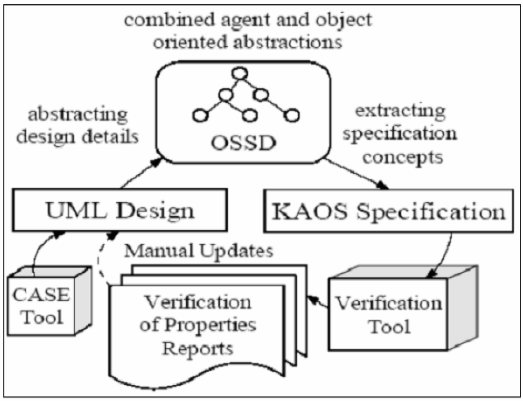


Figure 6.8. OSSD Approach

OSSD has the same objective as FADSE in producing high quality design; however, OSSD does not have a specific focus on security and it does not illustrate how to deal with UML semantic inconsistencies. Further, OSSD contemplates only on the design phase of software engineering with no strong focus on either requirements or implementation like FADSE.

Liu et. al. proposed a formalization of UML that defines both a UML model of requirements and another of design as a pair of class diagram and a family of sequence diagrams [46, 47, 48]. Both models of requirements and design are then given unified semantics. The approach then defines consistency between a design class diagram and the interaction diagrams and shows how the removal of inconsistency can be treated as a model refinement. Finally, the approach formally defines the correctness of UML model of design with respect to the model of requirements. This approach supports a "use case, step-wised and incremental development in building models for requirements analysis" [46].

In this formalization approach of UML, class models and use cases are used to capture requirements. The class model is relatively conceptual, which means that classes do not have methods and the associations among conceptual classes are undirected. The functional requirements are described by use cases and each use case is represented by a sequence diagram called a system sequence diagram. On the other hand, the design model consists of a design class model and a family of sequence diagrams. Classes in this class model now have methods and a method of a class may call methods of other classes. Therefore, the specification of these methods must agree with the object interactions in the sequence diagrams.

The formalization of both the requirements model and the design model allows for checking consistency between requirements and design [48]. However, the approach uses UML to build the requirements model, which suffers from the inherent inconsistencies and informality of UML that might result in building a low quality requirement model. Moreover, the formalization approach allows for eliciting and analyzing functional requirements while marinating the limitations of UML to capture nonfunctional requirements such as security.

Ledang and Souquieres proposed translating UML specifications to formal B specifications in order to rigorously analyze UML specifications via their corresponding B formal specifications [49, 50,

51]. This approach suggests a formalization of each UML construct as follows:

1. Use case translation to B: each use case is modeled as a B operation. To express in B the pre- and post-conditions of use cases, each use case and its involved classes are modeled in the same abstract machine. By structuring use cases, they are organized into levels. The use cases at level one corresponds to "user-goal" use cases. The use cases, which are the included cases of the ones at level one, are said at level two and so on. The bottom level of use cases is composed of basic operations of classes.
2. Modeling class operations: each class operation is modeled as a B operation in an abstract machine. As for use cases, the class operation and its involved data are grouped in the same abstract machine. In addition, the calling-called class operation dependency to arrange derived B operations into abstract machines is used.
3. Modeling state charts: this happens in two stages:
   - creating a B abstract operation for each event. In the B abstract operation, the expected effects of the event is directly specified on the data derived from class data related by the event. Consequently, an event and its related data are modeled in the same abstract machine;
   - implementing (or refining) the B operation in the first step by calling B operations for the triggered transition and actions.

Several kinds of analysis on UML specifications can be done after the translation to B such as the following:

- the consistency of class invariant;
- the conformity of object and state chart diagrams regarding the class diagrams;
- the conformity of class operations, use cases regarding the class invariant;
- the class operation calling-called dependency;
- the use case structuring.

The transformation of UML to B is close in methodology to the FADSE though it is different in its objective. That is, the main objective of translating UML to B is to allow for formal analysis of UML specifications through their corresponding B specifications. The other objective of the UML formalization is to use UML specifications as a tool for building B specifications, so the development of B specifications becomes easier. On the other hand, FADSE transforms KAOS requirements model to B in order to allow for the refinement of security requirements, which are critical, into an implementation with a high confidence that the generated implementation meets the security requirements and preserves the security properties. Unlike FADSE, the UML formalization falls short when it is applied to security requirements as UML does not have specific constructs to model security specifications.

Blackburn et.al. introduced the Test Automation Framework (TAF), which is a model-based verification approach that has been effective in detecting and correcting requirement defects early in the development process [52-55]. The TAF main objective is to reduce the manual test development effort and reduce rework though the integration of various government and commercially available model development and test generation tools to support defect prevention and automated testing of systems [52].

TAF supports modeling methods that focus on representing requirements, like the Software Cost Reduction (SCR) method, as well as methods that focus on representing design information, like SimulinkB or MATRIXx, which supports control system modeling for aircraft and automotive systems [53]. Blackburn uses model translation means to convert requirement-based or design-based models to a form understandable by T-VEC, the test generation component of TAF, to produce test vectors [54]. Test vectors include both inputs and the expected outputs along with requirement-to-test traceability information. T-VEC also supports test driver generation, requirement test coverage analysis, and test results checking and reporting. The test

drivers are then used to test the implementation functionalities during the testing phase [55].

Like FADSE, TAF derives test cases from the requirements model in order to verify the consistency and compliance between requirements and implementation and to provide sufficient traceability information. However, TAF is stronger in generating different types of test cases including unit testing, integration testing and acceptance testing while FADSE only generates acceptance test cases. This difference is due to the fact that TAF is a specialized testing framework while FADSE provides the generation of acceptance test cases as an extra verification step to ensure that the derived implementation meets the initial set of security requirements. It might be part of FADSE' future work to apply deeper analysis to the KAOS requirements model to generate other type of test cases.

Rationale

Like many high-assurance applications, there are cost and time reasons to focus the use of formal methodsto the key aspects of a system. For large software applications it can be cost-prohibitive to apply formal methods and many of these large systems have relevant security concerns. So, a compromise was made and formal methods were applied to software security aspects of the system in Formal Analysis and Design for Security Engineering. Further, the requirements elicitation and specification process is complex and the additional complexity of formulating these requirements using a formal method is overwhelming. So, Formal Analysis and Design for Security Engineering started with Van Lamsweerde's near-formal, goal-directed KAOS framework for identifying, elaborating, organizing, analyzing, and specifying security requirements [4, 2, 4, 3]. KAOS is a proven semi-formal requirements elaboration framework with an underlying formal infrastructure based on first-order temporal logic. From this base, Formal Analysis and Design for Security Engineering transforms the KAOS requirements specification into B, preserving the security properties so carefully expressed in KAOS. Then, using the B

refinement process, Formal Analysis and Design for Security Engineering systematically elaborate and refine the security requirements into a formal B design specification. To accommodate the fact that KAOS is not fully formal, take the idea of producing a test case suite based on the requirements model to help increase confidence through verification. Further, extending KAOS with more formality in a development platform like B allows for tracing security requirements at the various steps of development; that is during both design and implementation.

This section provides an argument for the rationale of the choices of KAOS and B to employ in Formal Analysis and Design for Security Engineering. The paradigm shift from the traditional approaches including semi-formal and formal methods to goalorientation has led the requirements engineering research community to argue about the effectiveness and usefulness of the new paradigm. The research effort in the goal-orientation domain has resulted in the major two frameworks namely KAOS and i* that represent the current state of the art in the domain. A number of case studies have been used to demonstrate the effectiveness of the goal-oriented paradigm and illustrate its strengths and limitation. The KAOS framework has been effectively demonstrated on more than 30 industrial projects that report outstanding success stories [2]. The i* framework has been demonstrated on a number of case studies, some of them are quite large systems. However, there is no quantitative analysis that precisely estimate the gains obtained from applying goal-oriented approaches over traditional approaches. As Van Lamsweerde and Mylopoulos, et. al. mentioned that preliminary empirical studies and their own experience with goal-orientation shows a strong potential in its application and its extensibility to formal methods [57, 23].

The above mentioned approaches that extend KAOS with extra formality to fill in the gap between requirements and later phases of development such as architecture and design show interest of the research community in the new goal-oriented paradigm. Researchers who extended KAOS either for architecture or design provided a

qualitative argument for their choice of the goal-oriented method. Their argument has been qualitatively based on the prominent features of goal-orientation that enables the enhancement of the current requirements engineering practice. Van Lamsweerde argued for the strengths of the goal-oriented KAOS framework that makes it suitable to requirements engineering since it overcomes the limitations of traditional semi-formal and formal approaches as mentioned above. There remain still some limitations to the approach in engineering requirements. The following paragraphs summarize the strengths and limitations of the KAOS framework. The following key points about goal orientation justify the choice of the goal-oriented KAOS framework for requirements analysis in Formal Analysis and Design for Security Engineering [57].

Goal-oriented modeling and specification takes a wider system engineering perspective; goals are prescriptive assertions that should hold in the system made of the software-to-be and its environment; domain properties and expectations about the environment are explicitly captured during the requirements elaboration process, in addition to the usual software requirements specifications.

1. Operational requirements are derived incrementally from the higher-level system goals they "implement".
1. Goals provide the rationale for the requirements that operationalize them and, in addition, a correctness criterion for requirements completeness and pertinence [58].
2. Obstacle analysis helps producing much more robust systems by systematically generating (a) potential ways in which the system might fail to meet its goals and (b) alternative ways of resolving such problems early enough during the requirements elaboration and negotiation phase.
3. Alternative system proposals are explored through alternative goal refinements, responsibility assignments, obstacle resolutions and conflict resolutions.
4. The goal refinement structure provides a rich way of structuring and documenting the entire requirements document.

5. Different levels of formality could be offered by the framework allowing one to combine different levels of expression and reasoning: semi-formal for modeling and structuring, qualitative for selection among alternatives, and formal, when needed, for more accurate reasoning.
6. Goal formalization allows requirements engineering-specific types of analysis to be carried out, like:
   - guiding the goal refinement process and the systematic identification of objects and agents [7, 59];
   - checking the correctness of goal refinements and detecting missing goals and implicit assumptions [11];
   - guiding the identification of obstacles and their resolutions [59];
   - guiding the identification of conflicts and their resolutions [2];
   - guiding the identification and specification of operational requirements that satisfy the goals [8, 60].

Van Lamsweerde argument of the goal-orientation characteristics that offer better handling of requirements analysis is supported by Mylopoulos, et. al. argument in [23]. Mylopoulos, et. al. argued that the adoption of the goal-oriented mindset is very important during requirements analysis because it deals with non-functional requirements and relates them to functional ones [23]. Further, goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives behind a software development project. Many of the requirements engineering research community have argued that capturing these interdependencies between organizational objectives and the detailed software requirements can facilitate the tracing of the origins of requirements and can help make the requirements process more thorough, complete, and consistent [23]. Mylopoulos, et. al. have strengthened their argument in favor of the goal-oriented paradigm by preliminary empirical studies showing that goal-oriented analysis can indeed lead to a more complete requirements definition than OOA techniques. Further, the authors' own experiences

in analyzing the requirements and architectural design for a large telecommunications software system confirm that goaloriented analysis can greatly facilitate and rationalize early phases of the software design process [23]. KAOS provides a graphical notation and semi-formal interface the hides the underlying formal infrastructure in order to increase the usability and applicability of the approach and decrease its cost of employment in industrial projects [2]. Van Lamsweerde stated that one of the frequently asked questions about KAOS when considered for use in industrial projects is about the minimal project size for which a KAOS approach is cost-effective. The answer of this question is that if the project is estimated to take 20 man days, the probability of a positive return on investment is quite weak as building a requirements model is time-consuming compared to the project size in this case [61]. However, a quantitative analysis on the consulting projects in which Van Lamsweerde and his team have applied KAOS shows that a typical requirements analysis of 4 to 8 man months has been needed. The typical duration of the requirements analysis phase is 3 months and the budget needed for it represents about 10% of the total project cost [2, 61, 62]. Figure 11 extrapolates the return on investment according to project size from the Van Lamsweerde's team experience and from the following hypotheses [61]:

- the cost of one developer is 0,6 k€ per day;
- the cost of one analyst is 1 k€ per day;
- about one development project over 2 experiments in which the cost overruns with about 189%;
- one of the two projects that failed is due to a requirements related problem;
- the cost of an ideal requirements analysis phase is estimated at 10% of the project cost with a minimum bound fixed to 30k€.

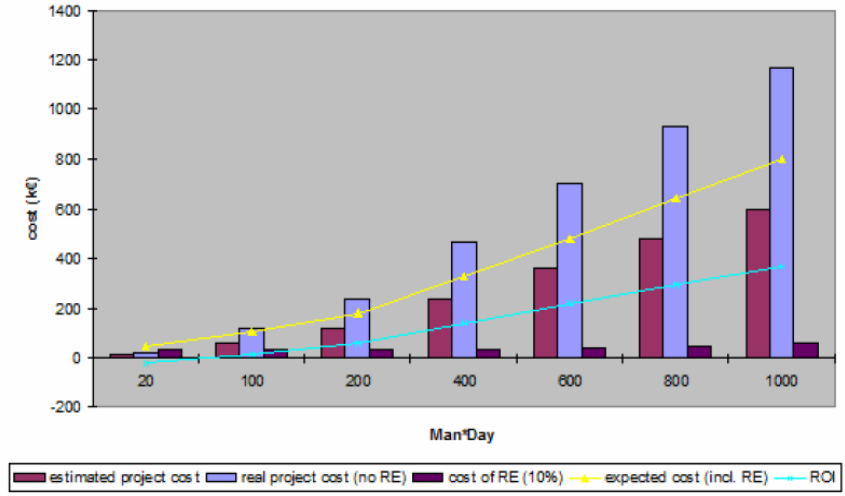


Figure 6.9: Return on Investment of KAOS Application According to Project Size [46]

Figure 6.9 shows that employing KAOS in a project for requirements analysis is cost-effective as soonas the project man power is more than 100 man days. *For medium-size and larger projects, the cost reduction is expected to be 30%* [61].

Figure 6.9 indicates that one of the limitations of the KAOS framework is that it is not cost-effective for projects whose size is less than 100 man days. In order to overcome this limitation, it is recommended that the company business develops a generic KAOS model once and customize it during the gap analyses made to compare the user requirements with what the package provides [22].

KAOS allows requirements engineers to use variable level of formality based on the criticality of the different parts of the software requirements. The variance in the formality level ranges from using the visual notation of KAOS to model goals to fully use first-order temporal logic to formally specify the goals, object invariants and operations pre/post conditions. Critical system aspects like security

requirements might be a good candidate to the employment of high level of formality for requirements analysis. Formality variance gives more flexibility to project managers to balance their tradeoff between effort and cost of formality.

The KAOS framework is capable of constructing near-complete and near-consistent requirements models. The word “near” has been cautiously used to describe the completeness and consistency of the KAOS requirements models since one of the KAOS limitations is that it is not capable of providing formal evidence of requirements completeness and consistency. Requirements models are generally characterized by being incomplete and inconsistent by nature [22] even with formal specifications. For example, in the electronic smart card case study used to demonstrate Formal Analysis and Design for Security Engineering, the integrity requirement on the messages communicated in the system has been missed though the case study was specified in the Z formal language. However, the KAOS framework provides a constructive procedure justifying its ability to reach reasonable completeness and consistency levels. First, KAOS specifies 5 completeness criteria that could be used by the requirements engineer to check and ensure the model completeness as follows:

1. A goal model is complete with respect to the refinement relationship if and only if every leaf goal is an expectation, a domain property, or a requirement.
2. A goal model is complete with respect to the responsibility relationship if and only if every requirement is placed under the responsibility of one and only one agent.
3. To be complete, a process diagram must specify:
   - the agents who perform the operations;
   - the input and output data for each operation.
4. To be complete, a process diagram must specify when operations are executed using trigger conditions.
5. All operations are to be justified by the existence of some requirements (through the use of operationalization links).

Second, KAOS provides a procedure for identifying all the possible obstacles (things that hinder goals'achievement) and conflicts (contradictions between requirements) during the obstacle analysis and resolution phase of the framework. In the security context, the KAOS security extension considers obstacles as possible security threats. The KAOS security extension provides a threat analysis mechanism to both formally or informally analyze possible threats and perform threat mitigation while building the security requirements model. Threat analysis results in the detection and resolution of security vulnerabilities very early in development [63]. Further, threat analysis allows for the anticipation of application-specific attack scenarios such as attacks on a web-based banking application that might result in disclosure of sensitive information about bank accounts or in credulous money transfer. My experience with using the obstacle analysis feature of the KAOS security extension has emphasized its effectiveness both in the demonstration of FADSE with the case studies and in the empirical study hold to validate Formal Analysis and Design for Security Engineering. For example, when applying Formal Analysis and Design for Security Engineering to the some system and comparing it to the Z specification and implementation of the same system, the obstacle analysis feature was effective in detecting threats to messages integrity. The Z specification, on the other hand, was not able to detect the same threats. This demonstrates that introducing KAOS for requirements analysis prior to formal design in Formal Analysis and Design for Security Engineering is not only enhancing the cost-effectiveness of applying formal methods, but also provides means to early detect security breaches and enhance the overall security of the software. Third, Van Lamsweerde indicated that spending reasonable effort on the construction of the requirements model enhances its completeness and consistency [2]. The more feedback sessions the analyst holds with the stakeholders to get their answers on open questions or to verify the completeness and consistency of the requirements model, the better the results obtained at the design and

implementation phases. The KAOS goal graph is a structure that could be communicated with the stakeholders who often have no technical background. Providing a structure that could be understood and communicated to stakeholders allows requirements engineers to get more thorough feedback from stakeholders to enhance the completeness and consistency of the requirements model.

The choice of B as a formal development platform for elaborating security design specifications assists in preserving security properties of requirements when design specifications are being derived. B has the notion of model refinement that allows for building a detailed model of design from an abstract model of requirements while preserving the security properties of the requirements model. The refinement mechanism in B provides a means for documenting design decisions and building forward traceability links from requirements to design. Hence, the links between artifacts are clear enough to provide traceability information that serves software maintenance activities, which might be performed after the software is fully developed. The use of a software model that stores design decisions and traceability links significantly improves the accuracy and completeness of impact analysis that is concerned with identifying the impact of a given change on the software product [65]. Moreover, B is based on set mathematics with the ability to use standard first-order predicate logic facilitating the integration with the KAOS security requirements model that is based on first-order temporal logic. Further, B is a mature formal method that has been successfully employed in industrial projects for long time. The availability of good tool support for the B development platform strengths the practicality and applicability of Formal Analysis and Design for Security Engineering.

Employing formal methods in Formal Analysis and Design for Security Engineering provides a reasonable approach to the challenge of developing secure software products with formal evidence of correctness [64]. Recognizing that formal methods reduce security risks but entails more cost, it is possible to justify this cost by applying

Formal Analysis and Design for Security Engineering only to security, which is a critical aspect of the system. Further, software systems that are evaluated for security at the Common Criteria (CC) EAL (Evaluation Assurance Level) 5, 6 and 7 need formal evidence assuring the security of the software. This makes software products developed using Formal Analysis and Design for Security Engineering securely compliant with CC higher levels.

Formal Analysis and Design for Security Engineering is a step towards the development of highly secure software. In a nutshell, Formal Analysis and Design for Security Engineering is a requirements-driven software engineering approach that derives design specifications from a set of security requirements modeled using KAOS security extension framework. The approach provides a secure software engineering methodology that effectively integrates KAOS security extension,which is characterized by the ability to formally build a complete, consistent and clear requirements model with the B method, which provides formal guarantees for the correctness of the system development. This research showed that KAOS is promising in that it could be extended with an extra step of formality in order to fully implement security requirements while preserving the security properties specified in therequirements model. Moreover, extending KAOS with more formality in a development framework like B allows for tracing requirements at the various steps of development; that is, during both design and implementation.

Formal Analysis and Design for Security Engineering starts with a set of security requirements that are being elaborated with the KAOS security extension to build a goal graph for the security requirements and derive the operations that achieve the goals. Formal Analysis and Design for Security Engineering makes use of the analytical capabilities provided with the goal graph to achieve two objectives. The first objective is to transform the KAOS operations derived to achieve the goals to B using means of the transformation scheme described below in order to construct an abstract B model that describes the initial

system state and its expected security behavior. The initial B machine is further refined using the B refinement mechanism to add more details while building the security design specifications. The second objective is to derive acceptance test cases from the goal graph outlining the different scenarios of security behavior that should be met by the derived B design and implementation specifications. This means that the goal graph is used to derive the initial abstract B model that will be further refined for design and implementation and to derive means to verify that the derived design and implementation meet the security requirements objectives through the acceptance test cases. The completeness and consistency of the derived implementation specifications are a function of the successful verification of the implementation against the acceptance test cases. The extra verification step that Formal Analysis and Design for Security Engineering provides through the derivation of test cases allows for detecting inconsistencies that might have been in the requirements model or in the process of design and implementation derivation. The test cases guarantee the same level of completeness and consistency of the requirements model since they are derived using a depth first search algorithm. The algorithm traverses the goal graph to generate sequence of calls to operations in the correct order that matches the semantics of the high level goals in the goal graph. The test cases provide means to detect possible security hazards that might result from inconsistencies either in the requirements model or in design. Formal Analysis and Design for Security Engineering is illustrated in Figure 6.10.

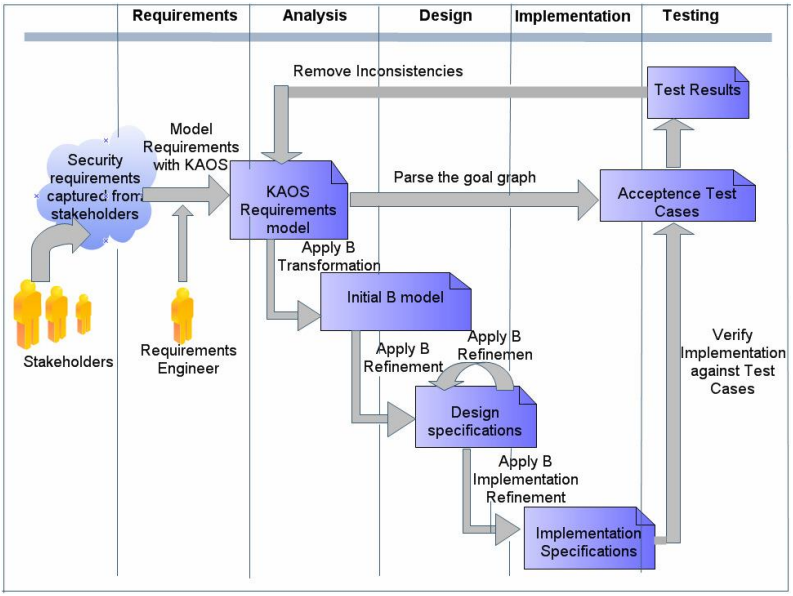


Figure 6.10. Formal Analysis and Design for Security Engineering

Formal Analysis and Design for Security Engineering provides means for transforming the security requirements model built with KAOS to an equivalent one in B using some transformation rules. The B model that has been transformed from KAOS representing security requirements is then refined using non-trivial B refinements that generate design specifications conforming to the security requirements. Each B refinement step prior to the implementation refinement reflects some design decision(s) added by the refining B machine to the refined B machine until implementation is obtained. The B formal method exhibits one of its prominent features of model refinement in allowing us to make our security design decisions with a proof of correctness that these decisions do not violate the constraints specified in the KAOS requirements model (operations pre/post conditions and entities invariants). This means that the development platform itself provides

constructs to reduce risks of introducing errors in development. After applying a number of refinement steps to the initial B model, an implementation refinement step, which is a special refinement step in B is applied.

Formal Analysis and Design for Security Engineering allows for deriving one artifact, which is design from another artifact, which is a requirements model using formal representation in B. The refinement mechanism in B provides means for documenting design decisions and building forward traceability links from requirements to design. Hence, the links between artifacts are clear enough to provide traceability information that serves the purposes of software maintenance activities that might be performed after the software is fully developed. The use of a software model that stores design decisions and traceability links significantly improves the accuracy and completeness of impact analysis that is concerned with identifying the impact of a given change on the software product [65].

The derived implementation specifications are then verified against the test cases that have been drawn from the KAOS requirements model. Our results have shown that the major two sources for problems encountered in implementation are inconsistencies either in the requirements model or in the design decisions made during the B refinement steps. The derived test cases are capable of detecting some inconsistencies in both the requirements model and design. Further, the ratio between the numbers of successful test cases and failed test cases can be used as exit criteria for security assurance and this is evidenced in the Common Criteria Evaluation Assessment Levels (EAL) 5, 6 and 7. The feedback loop established from the test cases results to the requirements model as indicated in the Figure 6.11 allows for detecting possible security hazards a priori to deploying the software system into production at which time “real” security threats might be encountered.

Formal Analysis and Design for Security Engineering is characterized with some features that make it more attractive to apply over other similar formal approaches. These features are either inherent

from the underlying approaches employed in Formal Analysis and Design for Security Engineering, namely KAOS and B or new in Formal Analysis and Design for Security Engineering. It addresses security-specific elements of software to bridge the gap between security requirements and their realization in design and implementation. Up to our knowledge, Formal Analysis and Design for Security Engineering is the first approach to address the gap between requirements and design for security requirements specifically. Formal Analysis and Design for Security Engineering is promising in being ready for wide applicability thanks to the strong tool support provided by the underlying technologies of KAOS and B. KAOS has a commercial tool called Objectiver [61] that has been commercially used in a number of successful industrial projects. B has been in the industrial market for a while with its most two famous commercial products namely AtelierB and the B-Toolkit [66, 67]. The availability of strong tool support strengths the practicality and applicability of Formal Analysis and Design for Security Engineering.

Formal Analysis and Design for Security Engineering takes KAOS with a further step of formality to derive design and implementation through transforming the KAOS requirements model to B. The initial B model obtained from the automatic transformation enforce the same security constraints specified in the KAOS requirements model modeled in the form of preconditions of the B operations that model the KAOS operations and the B machine invariants that model the KAOS entities invariants. The definition of these constraints in the initial B model forces the preservation of these constraints at the later B refinements steps that add more details to the initial B model to commit design decisions. The proof obligation facility provided with B method and that could be automatically generated using one of the commercial B tools allows software security developers to discharge the generated proof obligations to ensure correctness of development. Discharging the proof obligations formally proves that a refinement step in B does not violate the constraints of the more abstract B model being refined.

Formal Analysis and Design for Security Engineering provides an extra verification step to show the maintenance of security properties specified in the requirements model in the derived implementation specifications. This is achieved when deriving a set of test cases that are generated from traversing the KAOS goal graph. The test cases provide security developers with means to assure a reasonable level of completeness and consistency of their implementation with respect to the requirements model. Finally, one of the key merits of employing formal methods in Formal Analysis and Design for Security Engineering is the availability of sufficient traceability information that links requirements to design decisions giving better opportunities for more accurate and less vulnerable handling of changes to security specifications.

Transforming KAOS to B

The Knowledge Acquisition for autOmated Specifications requirements model is represented in the form of a directed-acyclic graph rooted at the very high level goal of the system and structured in multiple levels. Goals at each level refine the goals at the higher level. An example of a goal graph for the security requirements of the spy network system is illustrated as an example in Figure 6.12. The diamond shapes represent security goals while the rectangles with circular corners represent agents responsible to achieve leaf goal requirements.

The whole system is represented either as a single abstract B machine or multiple abstract B machines related to each other based on the size of the system. Each KAOS entity (passive object) is represented as a B machine included or seen by the system machine(s) and encapsulating its KAOS attributes and operations manipulating the attributes as follows:

- the entity attributes are variables in the equivalent B machine representing the state of the object;

- the B machine invariant is composed of the types of the attributes' variables (might be primitive types or types from other KAOS entities) and the domain invariant of the KAOS object presented in first-order predicate logic;
- each B machine representing an entity includes a set representing all its instances because B is not an object oriented language, rather it is instance-based. Entity attributes are represented as relations between the set of instances and the attribute types. This representation of the instances and their attributes allows for the use of the set arithmetic capabilities of B.

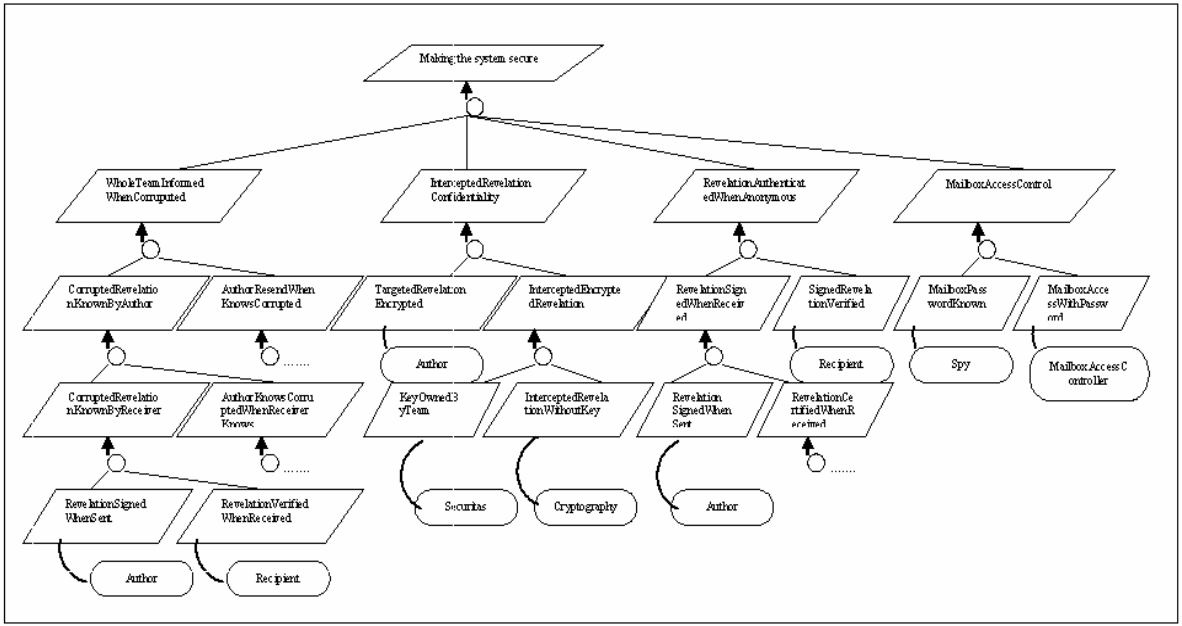


Figure 6.12: Spy Network KAOS Goal Graph

Each KAOS operation is represented as a B operation in the system machine and uses the KAOS entities either as parameters or return values. KAOS operations pre/post and trigger conditions are treated in the transformation as follows:

- pre-condition are directly mapped to a B precondition for the B operation since both KAOS and B preconditions are written in first order-predicate logic;
- post-condition has no equivalent construct in B. The operation specification and refinement should be responsible for enforcing the KAOS operation post-condition. This means that the post condition of the KAOS operation should be used in the operation specification, which is not part of the transformation. The transformation is only limited to building the requirements model that represent the KAOS model. It is the responsibility of the designer to fill in the body of each operation while taking into consideration the KAOS operation post-condition;
- trigger-condition has no direct mapping to the trigger condition in B. The trigger condition in KAOS is used to model the sequence of operation calls and the conditions that would lead to calling the operation. The trigger condition could be forced by the agents who are controlling the system runtime execution. Further, it is the responsibility of the agent calling the operations to prepare all the operations trigger conditions in a true state for the operation to be called.

Agents (active objects) are not directly transformed to KAOS since they are responsible for the runtime behavior of the system. Agents' behavior is modeled through the transformed KAOS operations. The acceptance test cases derived from the KAOS goal graph simulate the agent behavior in executing the KAOS operations that realized the requirements goals. KAOS goals are not transformed to B since their semantics is encapsulated in the KAOS operations that realize the leaf goals. The achievement of leaf goals through KAOS operations implies the achievement of the higher-level goals through the KAOS AND/OR refinement process. The KAOS goals and their sequence of refinement

are used to derive the acceptance test cases that are used to increase our confidence in the derived implementation specifications to be consistent and complete with respect to the requirements model. Further, KAOS goals in the goal graph are used to provide links for requirement traceability. Design decisions can be easily linked to higher-level goals using the goal graph structure that indicates which goals are realized by which operation (s). Traceability information plays a crucial role in accurately specifying the impact of applying a change to security specifications. KAOS domain properties are captured in B as invariants to the B machines that model the KAOS objects and as pre/post conditions to the operations that model KAOS operations. Objects invariants as well as operations pre conditions are preserved during the B refinement steps meaning that the domain properties are maintained throughout the software development lifecycle using the proposed approach. KAOS scenarios are not considered in the transformation scheme since scenarios are used as a vehicle security requirements engineer uses to assure the customer that all the security requirements are well understood and captured. The primary objective of the transformation from KAOS to B is to develop security design specifications that need to detail the implementation plan of the security requirements rather than focusing, as in the requirements analysis phase, on providing evidence that the customer can understand such as scenarios.

The rules of the transformation scheme from KAOS to B are summarized in Table 6.1.

Table 6.1. Transformation Rules from KAOS to B.

KAOS Constructs	B Constructs
KAOS object	B abstract machine
KAOS object attribute	B machine variable
KAOS object operation	B machine operation
KAOS object invariant	B machine invariant
KAOS operation	B operation
KAOS operation pre-condition	B operation pre-condition
KAOS agent	Agent behavior is modeled through the transformed
KAOS domain properties	Invariants of the transformed KAOS objects and pre-conditions of B operations

The transformation step of FADSE from KAOS to B can be automated using the Goal Graph Analyzer tool. The Goal Graph Analyzer tool is a Java tool that parses the KAOS goal graph represented in an XML format to generate the initial B model using the transformation rules of Table 6.1 and the acceptance test suite. The generated B model needs human in the loop to complement the B operations equivalent to KAOS operations and KAOS objects' operations with body specifications describing the abstract behavior of the operation to realize security requirements. The behavior need to be complemented since it is not specified in the KAOS model that only specifies what operations are needed to achieve security requirements rather than how these operations will achieve the requirements. To prove equivalency between the KAOS model and the initial B model resulting from the transformation, we can investigate two options; a formal mathematical proof and the derivation of acceptance test cases. Nakagawa and the team who developed the formal specification generator for KAOS wre consulted [38], that generates a VDM++ model from KAOS specifications about the feasibility of a formal mathematical proof of equivalency between KAOS and B. Nakagawa and his team have reported their experience in trying to develop the

mathematical proof as infeasible since the KAOS model is a requirements model involving undefined parameters making the proof very hard to develop within a reasonable timeframe. Moreover, the automation of the transformation reduces its error-proneness since automation rigidly applies the transformation rules. Further, the fact that both KAOS and B employ firstorder predicate logic to express system constraints and conditions facilitates the transformation and reduces the probability of an equivalency gap between the KAOS model and the transformed B model. The acceptance test case option has been more feasible to realize any errors that might result from the transformation scheme. The KAOS goal graph is used to derive the initial abstract B model that will be further refined for design and implementation and to derive means to verify that the derived design and implementation meet the security requirements objectives through the acceptance test cases. Our results have shown that the major two sources of problems encountered in implementation are inconsistencies either in the requirements model or in the design decisions made during the B refinement steps, but not in the transformation rules.

Derivation of Acceptance Test Cases

Formal Analysis and Design for Security Engineering provides an extra verification step to show the maintenance of security properties specified in the requirements model in the derived implementation specifications. FADSE derives a suite of acceptance test cases through traversing the KAOS goal graph. The test cases provide security developers with means to assure a reasonable level of completeness and consistency of their implementation with respect to the requirements model. The KAOS goal graph is rooted at the very high level goal of the system while leaf goals at the very bottom of the goal graph represent requirements. Each requirement is realized with an operation performed by an agent in the software-to-be. This means that the sequence of operations that need to be performed in order to execute a goal at a middle level of the goal graph can be identified using a depth first search (DFS) algorithm for the subgraph rooted at this goal. A DFS

algorithm with backtracking capabilities in order to eliminate the unnecessary paths from the search process has been chosen [68, 69]. Consider part of the goal graph concerned with the money exchange of electronic transactions for an electronic purse system illustrated in Figure 6.13. To test the achievement of the goal ExchangeMoney, the sequence of operation calls need to be generated using the DFS algorithm for the subgraph rooted at the ExchangeMoney node. Following the DFS algorithm illustrated in Figure 6.14 and its Java implementation in Figure 6.15, we can obtain the ExchangeMoney test case illustrated in Figure 6.16. The DFS algorithm searches the subgraph rooted at ExchangeMoney node to visit the operation nodes in the following sequence: DecryptMsg, SendMoneyValue, EncryptMsg, ReceiveMoneyValue

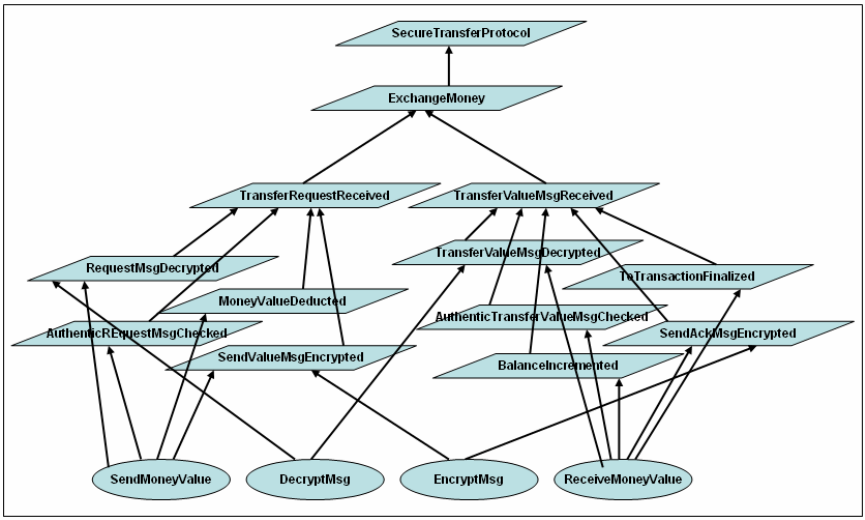


Figure 6.13. KAOS Goal Graph for Money Exchange of the Electronic Purse System

DSF (G,v)
Input: Goal graph G and a vertex
Output: Edges labeled as discovery and back edges in the connected component

For all edges e incident on v do
If edge e is unexplored then
W? opposite (v,e) // return the end point of e distant to v
If vertex w is unexplored then
- mark e as a discovery edge
- Recursively call DSF (G,w)
else
- mark e as a back edge

Figure 6.14. The Depth First Search Algorithm Used to Generate Acceptance Test Cases

```
//the arrays PreOrder and PostOrder have to be preallocated before calling the DFS algorithm.
import java.io.*;
public class search {
  static public int DFS(Graph G, int v, int[] PreOrder, int[] PostOrder) {

      int n = G.order();
      for (int i=0; i<n; i++) PreOrder[i] = PostOrder[i] = 0;
      // returns number of nodes reached
      countPair cnt = new countPair();
      doDFS(G, v, PreOrder, PostOrder, cnt);
      return PostOrder[v];
  }
  public static void main(String argv[]) {

   try {

      BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        Graph G = new Graph(input);
        int n=G.order();
            if ( n == 0 ) break;
        System.out.print(G.toStringAdjLists());
        int preOrder[] = new int[n];
        int postOrder[] = new int[n];
        GraphAlgs.BFS(G,0,preOrder);
        System.out.print(&#;``BFS (levelorder): &#;`);
        for (int i=0; i<n; i++) System.out.print(preOrder[i] + &#;`` &#;`);

        System.out.print(&#;``&#;\n&#;`);
        GraphAlgs.DFS(G,0,preOrder,postOrder);
        System.out.print(&#;``DFS (preorder): &#;`);
        for (int i=0; i<n; i++) System.out.print(preOrder[i] + &#;`` &#;`);

        System.out.print(&#;``&#;\n&#;`);
        System.out.print(&#;``DFS (postorder): &#;`);
        for (int i=0; i<n; i++) System.out.print(postOrder[i] + &#;`` &#;`);
        System.out.print(&#;``&#;\n&#;`);
      }

  } catch ( Exception e ) { System.err.println(&#;``Exception: &#;`+e); }

 } // main

} // class search
```

Figure 6.15. Java Code for the DFS

```
boolean exchangeMoneyTestCase(requestMsg,pd) {
```

```
decryptedReqMsg:=ElectronicPurse.decryptMsg(requestMsg);
if (decryptedReqMsg.content==null) return false;

encryptedValueMsg:=ElectronicPurse.
sendMoneyValue(pd.fromPurse,
                    decryptedReqMes);
if (encryptedValueMsg==null) return false;

decryptedValueMsg:=ElectronicPurse.
decryptMsg(encryptedValueMsg);
if (encryptedValueMsg ==null) return false;

ackMsg:= ElectronicPurse.
receiveMoneyValue(pd.toPurse,decryptedValueMsg);
if (ackMsg ==null) return false;
else return true;
}
```

Figure 6.16. Generated Test Case for MoneyExchange Goal

The generated test cases might need to be augmented by the software developers or testers to add more business domain-specific assertions. More meaningful messages might be displayed with the assertions to assist developers reasoning about failures of test cases and identifying sources of errors. Our results have shown that errors in the requirements model and/or design errors are the major sources of problems identified using the acceptance test cases. Blackburn proposed test coverage percentage and test results (number of successful and failed test cases) as measures for checking the quality of the software product [52]. The same measures could be used with Formal Analysis and Design for Security Engineering to evaluate the quality of the derived implementation, which is a function of the completeness and consistency of the requirements model. The acceptance test cases guarantee the same level of completeness and consistency of the requirements model since they are directly derived

from that model using the DFS algorithm. Therefore, the acceptance test cases are capable of identifying inconsistencies in the derived implementation with respect to the requirements model.

The derived test cases are automatically generated from the KAOS requirements model using the Goal Graph Analyzer tool that traverses the model to generate both the initial B model and the acceptance test cases. The acceptance test cases could be used as exit criteria for stakeholders to verify compliance between requirements and implementation. Further, the employment of formal methods in the derivation of both the acceptance test cases and the implementation allows for the assurance of target of evaluation constructed using Formal Analysis and Design for Security Engineering at Common Criteria EAL 5, 6, and 7 that mandate formal evidence of development. Clarke et. al. categorized testing into three categories namely unit testing, integration testing and acceptance testing [70]. Unit testing is concerned with assuring functionality on the module level while integration testing is concerned with assuring functionality when the various system modules are integrated. Clarke outlined selection criteria for paths that might reveal faults in software programs both for unit and integration testing [70]. However, up to our knowledge, there is no research work done in the area of identifying selection criteria for acceptance test cases that might have given more weight to some test cases over others in being able to reveal faults and inconsistencies.

FADSE Tool Support

Formal Analysis and Design for Security Engineering offers a wide range of applicability thanks to the integrated tool support provided by the underlying technologies of KAOS and B. KAOS has a commercial tool called Objectiver [61] that has been commercially used in a number of successful industrial projects. Van Lamsweerde has reported success stories of applying KAOS using the Objectiver toolbox to more than twenty enterprise software projects [2]. B has been in industry for a while with its most two famous commercial products namely AtelierB [66] and the B-Toolkit [67]. The availability of strong tool support

strengths the practicality and applicability of Formal Analysis and Design for Security Engineering and allows for its demonstration on large case studies. The KAOS requirements model created with Objectiver is exported in XML format. a Java tool called the Goal Graph Analyzer to parse the XML of the KAOS model and extract the necessary information required to build the initial B model and the acceptance test cases was built. The Analyzer produces two artifacts; 1) the initial B model representing the requirements model, and 2) the acceptance test cases. The software engineer needs to augment the B model with abstract specification of the B operations' body while the tester augments the generated test cases with some assertions and messages to enhance the usability of the test cases to produce more meaningful results. This inserts a human-in-the-loop to produce more useful output. The B-Toolkit is then used refine the abstract B machines to make design decisions and finally derive implementation specifications. Proof obligations are generated with each refinement step and the required proofs are discharged by the software developer in order to prove correctness of development. Figure 6.17 illustrates the tool support of Formal Analysis and Design for Security Engineering.

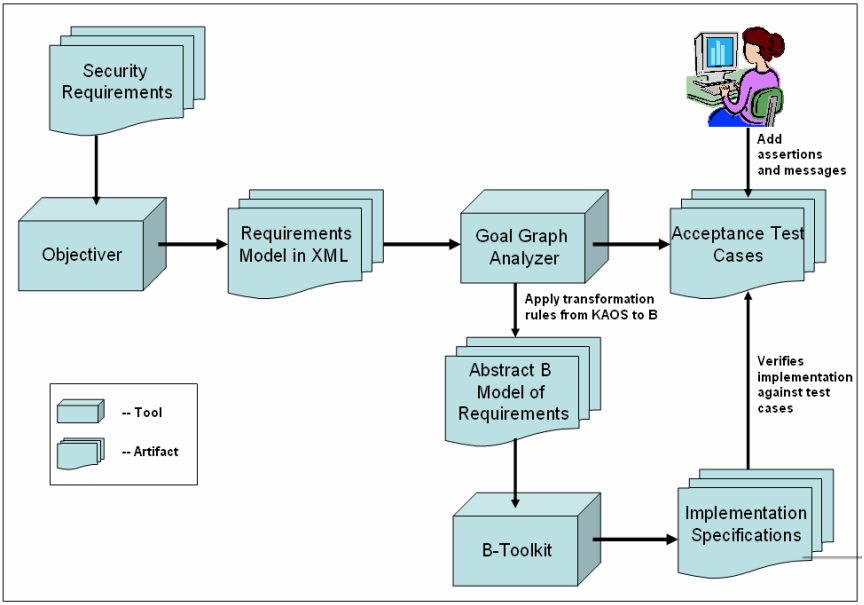


Figure 6.17. Formal Analysis and Design for Security Engineering Tool Support

6.3 A Formal Framework for Dependability and Resilience from a Software Engineering Perspective

Software engineering [71,72,73] aims at providing theories, methods and tools to allow for the production of *Information and Communication Technological* systems (ICT systems). Many adjectives can be used to qualify the production and the system built from different perspectives all of which are related to some so-called quality criteria. Two important areas of software engineering are model driven engineering and dependability. Model driven engineering [74, 75] is a field aiming at proposing methods and tools supporting the engineering of ICT systems for which models are extensively used. A model, in this

sense, is an abstraction of a real world entity that is of interest for the engineering of the ICT system under consideration. Models are defined using basic modelling elements and modelling elements' combination operators. Models are expressions (mainly textual or graphical) that comply with a modelling language (defined using a textual or graphical syntax). Examples of known modelling languages in software engineering are UML [76], BPMN [77], Statecharts [78] and of course all programming languages among others. The development of modelling languages is a continuous process that aims at tailoring the language to its targeted use.

The second field of interest, dependability, abstractly characterises the trustworthiness of a computing system. Dependability expresses informally the confidence that can be placed on services delivered (see http://www.dependability.org/). A well known informal conceptual framework has been proposed in [79, 80]. Here a taxonomy is proposed in which dependability concepts are organised into three categories: attributes (availability, reliability, safety, confidentiality, integrity, maintainability), threats (faults, errors, failures) and means (fault prevention, fault tolerance, fault removal, fault forecasting).

From an informal perspective means are used at specific points in the ICT systems's life cycle from its creation until its retirement in order to provide the targeted level for each of the attributes. In the last few years a significant amount of research and development has been dedicated to proposing languages, methods and tools to engineer dependable ICT systems. Using either explicit or implicit means that can be defined statically or dynamically in an autonomous or heteronomous manner. It is easy to understand and demonstrate that the dependability of an ICT system is a first class quality attribute. Even though the taxonomy, referred to above, represents a major improvement in the conceptual clarification of the concepts related to dependability, it is not sufficient for two main reasons. Firstly the taxonomy is not precise enough from a scientific point of view and secondly it has not been tailored to be used by the model driven engineering community. In order to gain precision, a first, simple and

commonly agreed upon approach is to provide a mathematical definition. To be compatible with the model driven engineering paradigm, this definition should be given using meta-modelling techniques.

Resilience in ICT systems, introduced around the seventies [81] has been most intensively used within the research community in the very few last years. By reviewing the important references we can notice that the word resilience, is used with a variety of definitions and at different levels [82, 83, 84]. With the same intention described previously, it is in the interest of ICT systems engineering community to derive a precise definition of resilience that can be also integrated into modelling languages.

With respect to the mathematical definition of concepts that are useful to understand dependability and resilience, we could use many fields (sets theory, mathematical logic, category theory, mathematical statistics and geometry among others) at different abstraction levels. Field selection should depend on the targeted exploitation of the formalisation. In this article, it is proposed to use algebra (mainly elementary algebra and basis of general algebra focusing on set theory and functions) since it is one of the mostly used mathematical fields by ICT systems engineers and scientists. Its concepts are easily mapped to the ones that exist in technological fields like programming languages and data bases. In addition, the concepts correspond to the terminology used in the natural language of document production throughout the ICT systems procurement life cycle (such as requirement document, verification and validation plans). Last but not least, the abstraction level chosen for the framework proposed in this article makes this choice appropriate. The intention is also that the framework should allow for being refined such that more detailed definitions of its concepts using different mathematical structures may be introduced in a consistent way with the framework. This should be considered at the

modelling language definition level. As an example, the dependability and resilience of a system might be dependent upon the metrics that can be defined based on probabilities, statistics, logical systems, model checking or test results. Thus these metrics should of course be present in the language used to model the dependable or resilient system.

In order to allow for the exploitation of the framework proposed in this article (called *DREF* for Dependability and Resilience Engineering Framework) in the modelling language definition, some integration techniques must be provided. To this aim and following the model driven engineering (MDE) perspective, a meta-model for the *DREF* framework was proposed to define. Further, the process for engineering new domain specific modelling languages (DSL) using a model driven engineering approach is presented. In this process, let us define how to exploit the *DREF* meta-model when introducing a new DSL. As proposed in this article, this can be done either by extending the existing DSL to integrate dependability and resilience support or upon creating the new DSLs.

In the following discussion, the formal conceptual framework for dependability and resilience is provided along with some basic illustrations. Then an approach for integrating the proposed framework into a modelling language definition is proposed and illustrated through a simple case study. A critical analysis and perspective section is then proposed to help understand the focus of this work.

Definition of a formal conceptual framework for dependability and resilience

In this section let us introduce all the concepts needed to define resilience. The main goal of these sections is to provide a precise mathematical set of definitions for all notions necessary to address resiliency.

Entities, Properties and satisfiability

The two first basic sets of the proposed conceptual framework (named *DREF*) are the sets of *entities* and *properties*. Entities are anything that is of interest to be considered. It might be a program, a database, a person, a hardware element, a development process, a requirement document, et cetera. Properties are the basic concepts to be used to characterise entities. It might be an informal requirement, a mathematical property or any entity that aims at being interpreted over entities.

Definition 1. The basic sets of the *DREF* conceptual framework, defined as disjoint subsets of a given universe *U* are:

6. PROPERTY $\in \wp(U)$ the set of all properties;
7. ENTITY $\in \wp(U)$ the set of all entities.

Remark 2.

a) $\mathrm{prop}, \mathrm{prop}_a, \mathrm{prop}_1 \ldots \mathrm{prop}_n$, (resp. $\mathrm{ent}, \mathrm{ent}_a, \mathrm{ent}_1 \ldots \mathrm{ent}_n$) will denote of *PROPERTY* (resp. of *ENTITY*);

b) Subsets of *PROPERTY* and *ENTITY* will be denoted by capitalization (e.g. *Prop*, *Ent*). Indexed notation might also be used.

The fact that a property is satisfied to some degree by an entity is defined as a function as follows:

Definition 3. Let *Prop*, *Ent* be sets of properties and entities; Satisfiability, *sat*, is a function such that:

$sat: Prop \times Ent \longrightarrow \mathbb{R} \cup \{\bot\}$

Remark 4.

a) We denote by $\mathrm{dom}(\mathrm{sat})$ the subset of $Prop \times Ent$ for which *sat* is defined;

b) $sat(prop, ent) = \bot$ represents the fact that the satisfiability of *prop* for *ent* cannot be determined (in addition to the fact that *sat* functions are partial functions). This is to allow one to differentiate the case where a satisfiability value is expected

but the satisfiability evaluation cannot be determined to the case where the satisfiability function is not expected;

c) We use $\mathbb{R}$ as co-domain of the satisfiability functions in order to cover all cases that would be necessary.

Example 5.

A simple satisfiability function could consider its co-domain partitioned using two arbitrary values (e.g. 1 and 0) such that:

8. $sat(prop, ent) = 1$ represents the fact that prop is "exactly satisfied" by *ent* (or *ent* "satisfies exactly" *prop*). In this context, 1 represents the nominal satisfiability;
9. $sat(prop, ent) = 0$ represents the fact that *prop* is "exactly unsatisfied" by *ent* (or *ent* "unsatisfies exactly" *prop*). In this context, 0 represents a tolerance threshold;
10. $sat(prop, ent) > 1$ (resp. ≤ 0) represents the fact that *prop* is "oversatisfied" (resp. "under satisfied") by *ent* (or *ent* "satisfies more (resp. less) than" *prop*);
11. in case, $sat(prop, ent) \geq 1$ (resp. ≤ 0), we can simply say that *prop* is "satisfied" (resp. unsatisfied) by *ent* (or *ent* "satisfies" prop). In this context, 1 (resp. 0) represents an acceptance (*resp*. rejection) threshold;
12. if $sat(prop, ent) \in [0,1]$ then we can consider that *prop* "is partly satisfied and partly unsatisfied" by *ent*.

For a strict Boolean satisfiability function only two satisfiability values would be accessible (e.g. 0 and 1). Thus only two notions would be used: satisfied or unsatisfied.

Let us consider that the notions of under and over satisfiability are necessary to represent the frequent informal situation for which an entity has more (resp. less) than the required property. The existence of an order relation between properties could represent the correlation of satisfiability between properties. When this order relation can be defined it could be useful that it is *strongly sat – compatible* as defined

below to indicate that a property greater than another must have its satisfiability value always greater than the one of the other for all the entities considered.

Definition 6. Let *sat* be a satisfiability function over *Prop* and *Ent* and $<_{prop}$ a partial order defined over *Prop* . $<_{prop}$ is said *strongly sat – compatible iff* the following property holds:

$\forall prop_a, prop_b \in Prop\ (prop_a <_{prop} prop_b) \Rightarrow (\forall\ ent\ \in Ent\ sat(prop_a, ent) < sat(prop_b, ent)\ or\ sat(prop_a, ent) = sat(prop_b, ent) = \perp)$

As stated in the introduction, the formal framework developed is intended to cover the current use of these concepts from an informal perspective (e.g. natural language words in expression), through the modelling expression in semi-formal notations up until mathematically based notation. Thus for each context, one should associate, at the right level, the concepts of the *DREF* framework to the existing concepts.

Example 7. Let us consider the context where mathematical logic is used to describe properties. In this case, entities will be logical structures (a set of logical structures is denoted *LogStruct* as a member of the power of all possible logical structures $\wp(LOGSTRUCT)$). Properties are logical formulae. Let *LSpec* denotes a set of logical formulae and *LStruct* a set of logical structures. Let *sat* be a satisfiability function such that:

$$sat: LStruct \times LSpec \rightarrow \mathbb{R} \cup \{\perp\} \quad \text{s.t. sat(Istruct, p)} = \begin{cases} 1\ if\ Istruct\ \vDash\ p \\ 0\ if\ Istruct\ \nvDash p \\ \perp\ else \end{cases}$$

Let us now define an order relation $<_l$ over *LSpec* s.t. $p <_l p'\ iff\ p \vDash p'$. Thus we have the following theorem:

Theorem 8. $<_l\ is\ strongly\ sat - compatible.$

Proof. $\forall Istruct \in LStruct\ sat(Istruct, p) = 1 \Rightarrow Istruct \vDash p$. Furthermore we know that $p <_l p' \Rightarrow p \vDash p'$. So, $Istruct \vDash p'$ and $so\ sat(Istruct, p') = 1$.

The same approach is possible if the properties would be algebraic specifications with a loose semantics and entities would be algebraic structures with a loose semantics [85].

Example 9. In this example, we consider an entity that corresponds to a first version of a software product line (SPL) platform developed in our laboratory [84] (see Table 6.2). This SPL platform (RF for REACT Framework) is used to derive Crisis Management Systems (e.g. car crash crisis, fire crisis in schools, pollution crisis). There are two properties to consider (see Table 6.3). The first states the compliance of the platform w.r.t. the service oriented architecture style [87] and the second the compliance of the entity RF with the definition of a SPL platform according to [88]. The decision depends on three stakeholders (lets call them observers as given in Table 6.4. A satisfiability function is given for each observer and for the REACT framework entity in Table 6.5. Each observer provides its satisfiability function. In this example, each property is evaluated using a discrete evaluation scale of naturals of [-5, +5] (with acceptance and rejection thresholds both set at 0).

Table 6.2. Entities.

1	RF	React Framework

Table 6.3. Properties.

1	SOA	is SOA oriented
2	SPL	is a SPL framework

Table 6.4. Observers.

1	NG	Nicolas Guelfi
2	BR	Benoit Ries

3	JL	Jerome Leemans

Subjectivity of satisfaction using observers and balancing

Satisfaction is not an objective concept as illustrated in example 9. To represent subjectivity, let us introduce the concept of "observer". Thus the satisfiability functions can be associated to observers.

Definition 10. The set of all observers is defined as a subset of a given universe U and is denoted *OBSERVER*.

Table 6.5. Satifiabillity function.

Satifiabillity function for RF		
Obs.	prop.s	sat.
NG	SOA	1
	SPL	-2
BR	SOA	2
	SPL	1
JL	SOA	4
	SPL	⊥

Remark 11. Let *o* be an observer, then sat_o denotes a satisfiability function for the observer *o*.

Now we can define the satisfiability function for a set of observers. At this point, no information is available in order to differentiate between observers. Of course, useful satisfiability functions need not only to allow for balancing observers but also properties. This is done later by introducing weights. in the meantime, we use the arithmetic average. Nonetheless, it must be reminded that the satisfiability value for each observer is free to be fixed.

Definition 12. Let Obs be a set of observers, and Sat_{Obs} a Obs – indexed family of satisfiability functions, then the satisfiability function for Obs is defined as:

$$sat_{Obs}: Prop \times Ent \longrightarrow \mathbb{R} \cup \{\bot\} \quad s.t. Prop = \bigcup_{o \in Obs} Prop_o \text{ and } Ent$$

$$= \bigcup_{o \in Obs} Ent_o \text{ and } sat_{Obs}(prop, ent)$$

$$= \begin{cases} \bot \text{ if } \exists_o \in \dfrac{Obs}{sat_o(prop, ent)} = \bot \\ \dfrac{\sum_{o \in Obs} sat_o(prop, ent)}{|Obs|} otherwise \end{cases}$$

Remark 13. We consider that satisfiability of a property p or an entity e can only be defined for a set of observers $if < p, e >$ belongs to the domain of each element of Sat_{Obs}.

Observers and properties might be balanced to reflect the fact that the global satisfiability function might be impacted differently by observers or properties. In many situations, the final satisfaction should take into account that observers are not always egual to each others and that some properties can count more than others. We thus introduce a notion of balancing in a generic way to cover, later, observers and properties too. As a first approach we consider weights (i.e. balancing values) as being positive and non-null values. To avoid considering an observer, It should be removed from the list of observers. A negative weight for an observer would mean that we should consider the opposite of all his judgements!

Definition 14. Let S be a set, then a balancing of S is defined as a function ω_S such that: $\omega_S: S \longrightarrow \mathbb{R}_+^*$.

Remark 15. We will consider an observer's balancing (e.g. ω_{Obs}) and a property's balancing (e.g. ω_{Prop}).

Definition 16. Let sat be a satisfiability function over $Prop$ and Ent, ω_{Prop} a balancing of $Prop$, then the global balanced satisfiability of sat, is denoted as $gsat_{\omega_{Prop}}$ and is such that:

$$gsat_{\omega_{Prop}} = \frac{\sum_{<p,e> \in dom(sat)} \omega_{Prop}(p) \times sat(p, e)}{\sum_{p \in Prop} \omega_{Prop}(p)}$$

Definition 17. Let Obs be a set of observers, Sat_{Obs} an Obs-indexed set of satisfiability functions, ω_{Obs} a balancing of Obs. A balanced satisfiability function for Obs, Sat_{Obs} and ω_{Obs} is denoted as $sat_{\omega_{Obs}}$ and is such that:

$$sat_{\omega_{Obs}}: Prop \times Ent \longrightarrow \mathbb{R} \cup \{\bot\}\ s.t.\ Prop = \bigcup_{o \in Obs} Prop_o \text{ and } Ent$$

$$= \bigcup_{o \in Obs} Ent_o \text{ and } sat_{\omega_{Obs}}(p,e)$$

$$= \begin{cases} \bot \text{ if } \exists_o \in \dfrac{Obs}{sat_0(p,e)} = \bot \\ \dfrac{\sum_{\substack{o \in Obs \\ <p,e> \in dom(sat_o)}} \omega_{Obs}(o) \times sat_o(p,e)}{\sum_{o \in Obs} \omega_{Obs}(o)} \text{ otherwise} \end{cases}$$

Since observers can be grouped we must analyze the properties of the satisfiability functions. We thus define the notion of coherence.

Definition 18. Let Sat_{Obs} be a Obs-indexed set of satisfiability functions. Sat_{Obs} is said to be coherent iff

$$(\forall o, o' \in Obs)(\forall p \in PROPERTY)(\forall e \in ENTITY) \neg (sat_o(p,e) = \bot \wedge sat_{o'}(p,e) \neq \bot) \wedge \neg (sat_o(p,e) < 0 \wedge sat_{o'}(p,e) > 0)$$

Definition 19. Let Sat_{Obs} be a Obs-indexed set of satisfiability functions. Sat_{Obs} is said to be homogeneous iff coherent and

$$(\forall o, o' \in Obs)(\forall p \in PROPERTY)(\forall e \in ENTITY)(sat_o(p,e) \leq 0 \wedge sat_{o'}(p,e) \leq 0) \vee (sat_o(p,e) \geq 0 \wedge sat_{o'}(p,e) \geq 0)$$

Example 20. If we consider the same case described in example 9, then we can define weights for observers and properties as given in Table 6.6. We can see that the observer *NG* (considered head of the project) has a weight of 3 compared to the weight 1 of *JL* (a master trainee). Concerning properties, all the observers share the same property weights which indicate that the software product line dimension (*SPL*) of the framework developed (*RF*) is twice more important that the software service oriented one (*SOA*).

Thus we have the following global balanced satisfiability values for RF:

- for the observer NG it is $gsat_{NG} = -1$;
- for the observer BR it is $gsat_{BR} = 4/3$;
- for the observer JL it is $gsat_{JL} = 4$;
- for the set of observers $Obs = \{NC, BR, JL\}$, we have a global balanced satisfiability of $\frac{3 \times gsat_{NG} + 2 \times gsat_{BR} + 1 \times gsat_{JL}}{3+2+1} = \frac{11}{18} \approx 0.61$.

Of course, since JL is not capable of evaluating the SPL property over the RF entity, the family of functions is not domain-homogeneous. Thus the computation of the global satisfiability is biased. If we compute the maximum set of domain-homogeneous observers from Obs (i.e. $Obs_h = \{NG, BR\}$), then the global satisfiability of Obs_h is $\frac{-1}{15} \approx -0.06$ and thus we move from a positive (above the acceptance threshold) to a negative value (i.e. below the rejection threshold). This result indicates that the scale chosen has changed from an acceptance status to a rejection status.

It must be noticed that the global satisfiability function is not intended to be the only or the primary satisfiability information to be used when exploiting the $DREF$ framework. Doing so would imply the exclusion of all approaches that would be required to handle specific properties or observers since they would be hidden in the global weighted average computation.

Table 6.6. Weights and Satisfiability function.

Weights and Sat for RF				
Obs.	w.o.	prop	w.p.	sat.
NG	3	SOA	1	1
		SPL	2	-2
BR	2	SOA	1	2
		SPL	2	1
JL	1	SOA	1	4
		SPL	2	⊥

Balancing is an important concept for explicit definition of priorities. In all the engineering projects we have been involved in (from 10k€ to 3 M€) implicit or explicit prioritization, ordering, or balancing of properties were incorporated by partners (i.d. observers), themselves implicitly or explicitly weighted. Nevertheless, it is true that the current practices have difficulties in explicitly stating those weights. This is also the case for explicit modeling of some types of faults, especially the ones that are of all the following types: development, internal, human made, software level, non malicious, non deliberate, incompetence due and persistent. We believe, nevertheless, that accurate scientific models should be able to handle them. Further work is required in experimenting with methodologies in which the weights can be efficiently incorporated.

Change, Evolution Axis and Correlations

The terminology used in the fields related to Information and Communication Technological (ICT) systems, quite frequently incorporates the following keywords: change, evolution, adaptation, variation, modification, transformation. If we focus on a program as an entity, then a program change could refer to a new version of the program seen as a sequence of lines of code, or it might refer to the change of some program "status" defined using specific "state variables". One can easily see that those two interpretations of program change are fundamentally different.

Considering the notion of change seems logical since we are currently considering ICT systems and humans as being entities whose existence (i.e. definition) seems to change with time (at least). A first simple approximation, then, would be to define change as the difference between two definitions of two entities distributed over a common evolution axis. As an example, let consider p_1 as a simple imperative program sorting a list of integers using the bubble algorithm and p_2 using a guick sort algorithm. If we consider p_1 and p_2 as

comparable entities then the change from $p_1$to p_2 should represent the difference between the two programs. The way of defining the difference is thus fundamental. A simple very informal definition could be: "p_1 differs from p_2 by the type of sorting algorithm used". A more precise definition could be provided by the use of a term rewriting function $rewrite$ that would rewrite a bubble sort imperative program on to a guick sort based imperative program. In this case $p_2 = rewrite(p_1)$, will be used to define the difference between the two programs in terms of all the modifications made to p_1 to reach p_2

In case of a program status change, a program change would be defined as any modification to any of the predefined state variables constituting the program status. If the program status of p_1 is defined by the status of a local variable containing the list of integers to be sorted then p_1 and p_2 could not be considered as eguivalent at any point in the evolution axis except at initial and final evolution points. In this last case, the difference between the two programs is expressed as a difference between two lists of integers. Thus the definition of the evolution axis is a mandatory preliminary step to allow an entity's comparison.

Definition 21. An evolution axis is a set of values that are used to index a set of entities or a set of properties.

Remark 22. The intention is to allow for comparison of entities relative to an evolution axis. Concerning ICT systems, the commonly used axes are the time axis (that can be considered as discrete or continuous) related to system's versioning or related to system status. If we consider software product lines then one evolution axis can be related to variants.

Example 23. If we consider the REACT framework (entity RF), we can have three versions of the framework. One of them has an added service discovery mechanism based on a service registry, and the third has provided service orchestration of reusable modules corresponding to different types of crisis management scenarios belonging to different crisis types explicitly provided in the framework. Let us introduce an

evolution axis representing the three successive versions of the REACT framework. The evolution axis is then the set $\{v_1, v_2, v_3\}$ and it concerns the entity RF. We will then have three values on this evolution axis: $RF_{v1}, RF_{v2}, RF_{v3}$. It is important to notice that those values are strictly ordered. This is mandatory to consider when addressing resilience.

Example 24. Another more complex illustration can be made in the context of the relationships between requirements and implementation. A classical situation in a system's life cycle is the fact that the correspondence between requirements and realisations is not always optimal. Of course the problem is to define what the criteria are to evaluate and order these correspondences. A simple approach could be to compare the set of user functionalities described in the requirement document and the set of functionalities supported by the system. This corresponds to a comparison between the requirement document obtained at the analysis level and a reverse engineering of the requirement from the implemented system. In this case, one can define two evolution axes, one for the implemented system and one for the reguirements. We thus have the properties' evolution axis $Prop_{evo_1} = \{v_1, v_2, v_3, v_4\}$, and the system's implementation evolution axis $Ent_{evo_1} = \{v_1, v_2, v_3, v_4\}$. We have four values for the properties $Prop = \{req_{v1}, req_{v2}, req_{v3}, req_{v4}\}$ and three values for the system entities $Ent = \{sys_{v1}, sys_{v2}, sys_{v3}, sys_{v4}\}$. The satisfiability function would then evaluate the adequacy of requirements vs implementations. This function could be defined as a percentage of the functionalities of requirements covered by the system. In this case, we could have the satisfiability functions defined in the tables 6.7, 6.8 and 6.9. Depending on the evolution processes definition and coordination for the requirements and for the implementations, the set of adequacies to be evaluated is defined characterising the domain of the satisfiability function.

Table 6.9. Weights and Satisfiability function for sys-v1.

Weights and Sat for sys-v1

Obs.	w.o.	prop.	w.p.	sat.
NG	1	req-v1	1	75
		req-v2	1	60

Table 6.8. Weights and Satisfiability function for sys-v2.

Weights and Sat for sys-v2				
Obs.	w.o.	prop.	w.p.	sat.
NG	1	req-v2	1	80

Table 6.9. Weights and Satisfiability function for sys-v3.

Weights and Sat for sys-v1				
Obs.	w.o.	prop.	w.p.	sat.
NG	1	req-v3	1	85
		req-v4	1	90

Nominal Satisfiability and Requirements

In order to formalize the notion of quality, we must introduce the concept of a nominal satisfiability function and the concept of requirement.

Definition 25. A nominal satisfiability function is a satisfiability function used to represent the expected satisfiability and to allow comparative evaluation w.r.t. to any satisfiability function that is provided.

Definition 26. A requirement for a set of entities, Ent, is a set of properties, Req of $PROPERTY$, together with a nominal satisfiability function, $nsat$, such that $dom(nsat) =< Req, Ent >$

Remark 27. When not provided, the default nominal satisfiability function is such that $\forall < p, e > \in dom(nsat)\ nsat(p, e) = 1$.

Definition 28. Let $rqt =< Req, nsat >$ be a requirement for a set of entitles, Ent, and sat a satisfiability function, sat is said to satisfy $rqt\ iff\ \forall < r, e > \in dom(sat)\ \ sat(r, e) \geq nsat(r, e)$.

Example 29. If we consider the REACT framework (entity RF), we have three versions of the framework as described in example 23. The tables 6.10, 6.11 and 6.12 given below describe the values of the satisfiability functions for the three observers and the same properties.

In this example, we have a nominal satisfiability function $nsat(r, e) = 1$ for all versions of the RF entity and for the two properties. The evolution of the global satisfiability function is the following:

gsat('RF – v1') ≈ 0.61

gsat('RF – v2') ≈ 1.22

gsat('RF – v3') = 3

The evolution of the global balanced satisfiability function is the following:

$gsat_{wObs}$('RF – v1') ≈ – 0.06 (in case of a selection of homogeneous observer set)

$gsat_{wObs}$('RF – v2') ≈ 1.22

$gsat_{wObs}$('RF – v3') ≈ 2.93.

This means that RF − v1 and RF − v2 satisfy the requirements if we do not take into account the balancing (gsat is greater than the nominal satisfiability defined at 1) and that only RF − v3 satisfies the requirements if we consider balancing (the two others – 0,06 and 0:86 are lower than 1).

Table 6.10. Weights and Satisfiability function for RF-v1.

Weights and Sat for RF-v1				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	1
		SPL	2	-2
BR	2	SOA	1	2
		SPL	2	1
JL	1	SOA	1	4
		SPL	2	nil

Table 6.11. Weights and Satisfiability function for RF-v2.

Weights and Sat for RF-v2				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	3
		SPL	2	-1
BR	2	SOA	1	3
		SPL	2	1
JL	1	SOA	1	5
		SPL	2	2

Figure 6.18 gives a general graphical representation of the satisfiability functions (Y axis) for all the observers (colors) over the properties (Z axis) and the REACT Framework entity evolutions (X axis). For example, we can see the progression of the satisfiability for NG for each property and along the evolutions of RF.

Table 6.12. Weights and Satisfiability function for RF-v3

Weights and Sat for RF-v3				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	4
		SPL	2	1
BR	2	SOA	1	3
		SPL	2	5
JL	1	SOA	1	4
		SPL	2	3

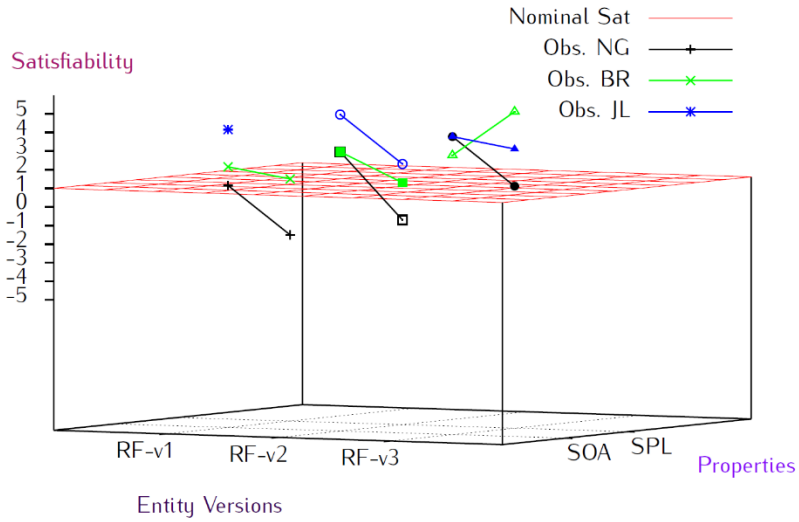


Figure 6.18. RF Evolution Satisfiability Function represented in 3D

Tolerance, Preservation, Improvement and Degradation

In order to address the main concepts of dependability, we propose to define four basic concepts in our formal framework: tolerance, preservation, improvement and degradation. We consider that there is no universal notion of quality but rather that quality is a subjective notion relative to a set of properties that represent the expectation on a specific entity.

Definition 30.

A tolerance threshold is defined as a satisfiability function. Tolerance threshold functions are used to represent the lower bound that defines the tolerance margin for satisfiability functions w.r.t. a nominal satisfiability function.

Remark 31.

Tolerance threshold functions will be denoted as tolsat.

The tolerance margin is the space between nsat and tolsat (see Figure 6.19).

Intolerance is characterised by nsat = tolsat.

Definition 32.

A preservation is defined as constancy in a satisfiability function w.r.t. an evolution axis.

Definition 33.

An improvement is defined as an increase in a satisfiability function w.r.t. an evolution axis.

Definition 34.

A degradation is defined as a decrease in a satisfiability function w.r.t. an evolution axis.

Remark 35.

Preservation (resp.improvement, degradation) might be observed relative to a nominal satisfiability and tolerance threshold in order to

discriminate between different types of preservation (resp.improvement, degradation). As an example, an improvement causing a satisfiability function to go from below to above the tolerance threshold would be characterised as a failure reducing improvement.

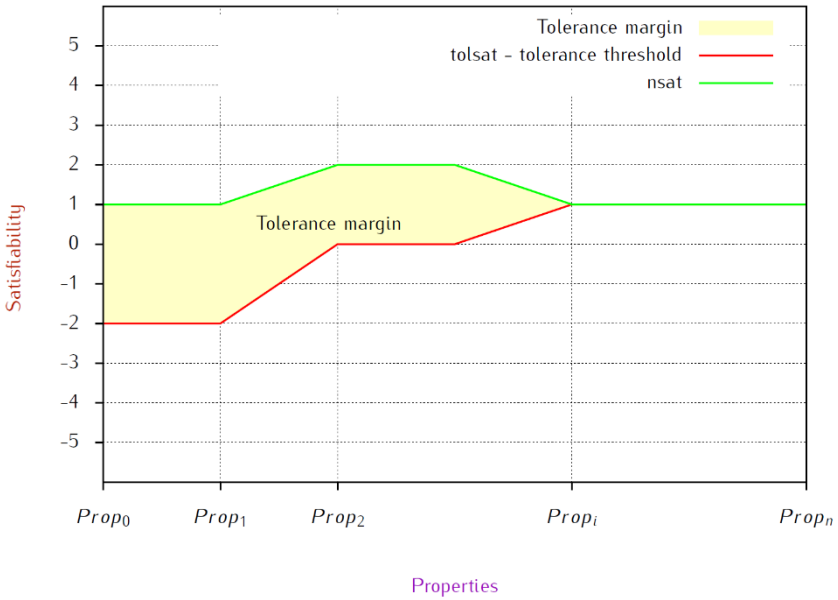


Figure 6.19. Tolerance threshold.

Example 36.

We consider the REACT framework and the three versions of the framework described in example 23 for which the satisfiability function is drawn in 2D in Figure 6.20. Considering the nominal satisfiability function nsat, we notice that for almost ail pairs of observers (NC,BR,JL) and properties (SOA, SPL), we have continuous improvement along the entity's evolution axis representing the React Framework versions (RF_{V1}, RF_{V2}, RF_{V2}). For < BR,SOA >, we have an

improvement from RF_{V1} to RF_{V2} and a preservation from RF_{V2} to RF_{V2} . For < JL,SOA >, we have an improvement from RF_{V1} to RF_{V2} and a degradation from RF_{V2} to RF_{V2}. If we now consider the varying nominal satisfiability function, nsat', we observe that there is no improvement but preservation for < BR,SOA > (and < NC,SPL >,< JL,SOA >) RF_{V1} to RF_{V2}.

Tolerance and Failure

Based on the previous definition of tolerance margin, we now address the notions of tolerance and failure. Both concepts are related to the satisfaction of a property over an entity given a nominal satisfiability and tolerance threshold functions.

Definition 37.

Given sat, a satisfiability function, and tolsat, a tolerance threshold, a failure is defined as a tuple < r,ent >ϵ dom(sat) $\cap$ dom(tolsat) such that: sat(r, ent)$\leq$tolsat(r, ent).

Remark 38.

1. We will write fail(r, ent) to denote the failure < r, ent >

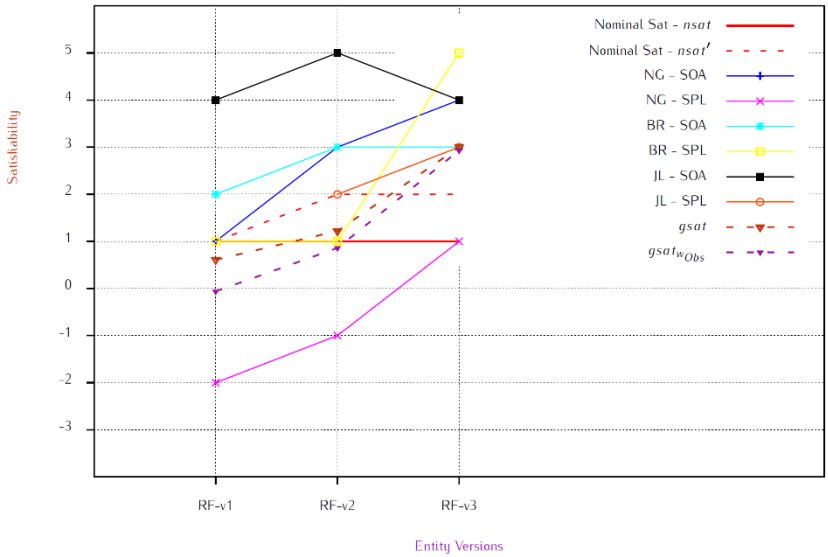


Figure 6.20. RF Evolution Satisfiability Function 2DGraph.

2. The fact that 3 < r, ent > ϵ dom(sat)/sat(r, ent) < nsat(r, ent) is considered as a degradation of requirement satisfaction untif sat(r, ent) ≤ tolsat(r, ent), a situation in which we have a faifure.

3. An infinite degradation is a tupie < r,ent > that is a faifure for which there is no entity ent' onward in the evofution axis of ent such that sat(r, ent') > tolsat(r, ent).

4. If we want to represent degradation modes it is sufficient to provide a strictly ordered family of tolerance thresholds that partition the tolerance margin into tolerance spaces, that correspond to the different degradation modes.

Definition 39.

Let sat be a satisfiability function, nsat a nominal satisfiability function and tolsat a tolerance threshold. Then a tolerance is defined as a tuple < r, ent > such that:
<r,ent> ϵ dom(sat) ∧ <r,ent> ϵ dom(tolsat)∧<r,ent> ϵ dom(nsat) ∧ sat(r, ent) < nsat(r, ent) ∧ sat(r, ent) > tolsat(r, ent).

Thus in order to allow the characterisation of fault-tolerance at the requirement level, we need to introduce the notion of a requirement with fault-tolerance .

Definition 40.

Let < Req,nsat > be a requirement and tolsat a tolerance threshold, then a requirement with fault-tolerance Reqrr is defined as a tuple <Req, nsat, tolsat >.

As an example, let us consider a set of comparable[11] entities, $p_{t1},\ldots,p_{tn}$, as being n number of changes of a program, p, over the time evolution axis. Let sat, nsat, tolsat be satisfiability, nominal satisfiability and tolerance threshold functions whose values are sketched in Figure 6.21. We have two failures between P_{tj} and P_{tj+r} which are the situations where sat (in black) falls below the tolsat tolerance threshold (in red). From P_{tj+r} till P_{tk} sas is within the tolerance limits characterised by having sas between nomsat (in green) and tolsat. Of course we must precise the context of the sat, nsat, tolsat functions which will mainly be dependent on the properties and observers and which should be coherent for the three functions in order to allow a precise interpretation of the tolerance, failure and preservation situations. Positive (resp. negative) variations of sat correspond to improvement (or degradation) w.r.t. satisfiability and constancy (from P_{t0} tp P_{t1} and from P_t, onward) corresponds to preservation.

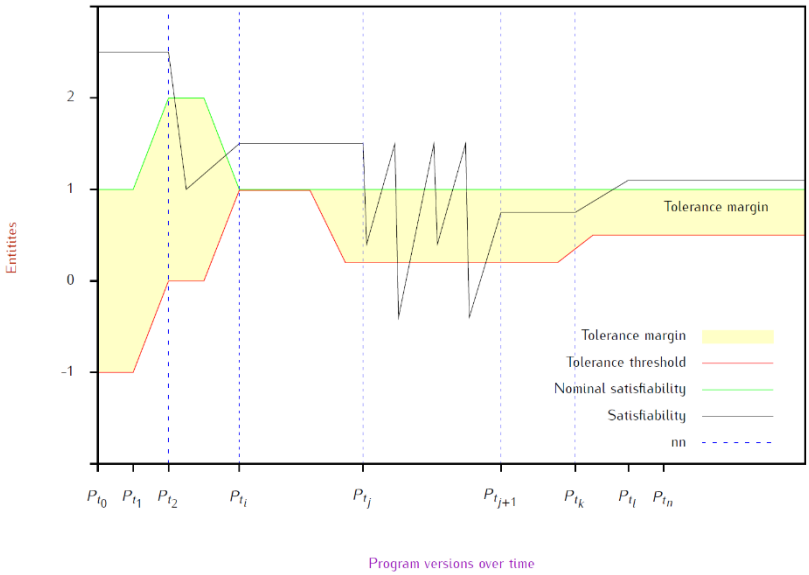


Figure 6.21. Tolerance threshold

3. Resilience as change for improvement

We define the general concept of resilience intuitively as a property of an evolution process that is considered to improve capabilities thus avoiding failures and reducing degradations. Roughly, it is the existence of a change toward improvement that reduces failures and tolerance needs. This fact implies that at least one evolution axis exists for resilience definition. Another evolution axis might be introduced as a refinement. The consideration of additional evolution axes can be useful to introduce different evolution types and to study their correlation and impact on resiliency. Different types of resilience might be introduced that depend on the evaluation of the expected reduction of failures and tolerances that are induced by the evolution. The

properties used for the evaluation of the satisfiability function define observation points. Thus an observation axis could also be introduced to classify the observation points used for the evaluation of failures and degradations and their change over the chosen evolution axes. The explicit concep¬tualisation of evolutions and observations is a fundamental task to allow for good dependability and resilience evaluation.

The next definitions are fundamental in our framework. The first introduces: tolmax as the maximum possible level of tolerance needed. It is defined based on the differences between the expected lowest acceptable satisfiability levels (in Figure 6.23 it is the yellow surface); stol represents the total quantity of tolerance deduced from the effective satisfiability all along the evolution axis (in Figure 6.23 it is the green surface); /to/ is the proportion of the two previous quantities. The second definition introduces a variation of the required tolerance levels between two evolutions (successive or not). The third definition focuses on failures and introduces: 1) a failure level that indicates the failure grade for a given property and entity evolution (in Figure 6.23 it is the distance between the red curve and any point of an entity version curve that goes below it); 2) qfail represents the number of times an entity is failing (goes below the tolerance threshold) all along its evolution axis; and 3) sfail quantifies the level of failure as for stol. The fourth definition introduces the variation quantities in terms of number of failures (Δqfail) and failure level (Δfail).

Definition 41.

Let ent_v be an entity, and ev = {1,…,m} be an evolution axis for ent_v. Let sat be a satisfiability function defined over entv and over a set of properties Prop, nsat a nominal satisfiability function and tolsat a

tolerance threshold defined over entv along the evolution axis. The notions of maximum tolerance ($tolmax^v$), cumulative tolerance ($stol^v$) and tolerance level ($ltol^v$) are defined as follows:

$$tolmax^v = \sum_{\substack{t \epsilon ev \\ p \epsilon Prop}} (nsat(p, ent_v^t) - tolsat(p, ent_v^t))$$

$$stol^v = \sum_{\substack{t \epsilon ev \\ p \epsilon Prop}} \begin{pmatrix} Max(sat(p,ent_v^t),(p,ent_v^t)) \\ -Max(sat(p,ent_v^t),(p,ent_v^t)) \end{pmatrix} (0 \; if \; Prop = \emptyset)$$

$$ltol^v = \frac{stol^v}{tolmax^v}$$

Definition 42.

Let enti and entj be two entities both evolving along an evolution axis ev = {1,…,m}. Let sat be a satisfiability function defined over the cited entities and over a set of properties Prop, nsat a nominal satisfiability function and tolsat a tolerance threshold defined over the entities along the two evolution axes. The notion of tolerance variation ($\Delta tol_{i,j}$) induced by the evolution from ent_i to ent_j is defined as follows:

$$\Delta tol_{i,j} = ltol^l - ltol^l$$

Definition 43.

Let entv be an entity, let ev = {1,…,m} be an evolution axis for the entity entv . Let sat be a satisfiability function defined over entv and over a set of properties Prop, nsat a nominal satisfiability function and tolsat a tolerance threshold defined over enti along the evolution axis. The notions of local failure ($fail^v$), cumulative failure quantity ($qfail^v$) and cumulative failure level ($sfail^v$) are defined as follows:

$$fail^v : ev \times Prop \rightarrow R / fail^v(t,p) = \begin{pmatrix} tolsat(p,ent_v^t) \\ -Min(sat(p,ent_v^t),(p,ent_v^t)) \end{pmatrix}$$

$$qfail^v = |\{< t,p > / fail^v(t,p) > 0\}|$$
$$sfail^v = \sum_{\substack{t \epsilon ev \\ p \epsilon Prop}} fail^v(t,p) \, (0 \; if \; Prop = \emptyset)$$

Definition 44.

Let enti and entj be two entities attached12 to a common evolution axis ev = {1,…,m}. Let sat be a satisfiability function defined over the cited entities and over a set of properties Prop, nsat a nominal satisfiability function and tolsat a tolerance threshold defined over the entities along the two evolution axes. The notions of failure level variation (faili;j) and failure quantity variation (qfaili;j) induced by the evolution from enti to entj is defined as follows:

$$\Delta fail_{i,j} = sfail^i - \; sfail^j$$
$$\Delta qfail_{i,j} = qfail^i - \; qfail^j$$

The next definition then, provides a first basic definition of resilience as a property over two entities belonging to a common evolution axis. $resil^T$ iff the tolerance level has decreased; $resil^F$ iff the number of failures as decreased; and $resil^{TF}$ if both previous properties are true.

Definition 45.

Let ent be an entity evolving along the evolution axis ev = {1,...,n} (the generative axis). Let sat be a satisfiability function defined over the cited entities and over a set of properties Prop, nsat a nominal satisfiability function and tolsat a tolerance threshold defined over the entities along the evolution axis. Let i,j ϵ ev, the properties of T-resilience.F- resilience and TF-resilience (noted $resil^T_{i,j}$, $resil^F_{i,j}$, $resil^{TF}_{i,j}$) are defined as follows:

- $resil_{i,j}^{T} is\ true\ iff\ \Delta tol_{i,j} > 0$
- $resil_{i,j}^{F} is\ true\ iff\ \Delta qfail_{i,j} > 0$
- $resil_{i,j}^{TF} is\ true\ iff\ resil_{i,j}^{T} > 0$

The definition of TF-resilience is a strict definition of resilience since it requires that both the level of tolerance and the number of failures if any are reduced.

Remark 46.

In case we want to address property (resp. properties set) specific resilience, we must define the set Prop used to compute the different types of resilience. By default, we will consider Prop as the set of properties over which sat is defined.

To illustrate these definitions, let us consider ev1 = versions = {1,…, n} be the n evolutions steps of an entity ent over the versioning axis and obs = times = $\{k_0,\ldots,k_m\}$ the m + 1 observation points for ent corresponding to a time axis. ent_i^k represents the entity at version i and time k. In Figure 6.22, we provide the graph of the satisfiability functions for ent_i ; ent_{i+1} and ent_{i+2} for the time observation points from k_0 to k_{10}. We consider only one property, a constant tolsat function of 0 and a constant nsat function of 1. If we compute fail and tol for the three evolutions enti; ent_{i+1} and ent_{i+2}, we obtain the figures given in Table 6.13 and in Table 6.14. We notice that the tolerance level ltol decreases from 0.86 to 0.23, which corresponds to a tol of 0.63. From a graphical point of view, we observe that through the two evolution steps, the total level of tolerance is constantly reduced (this is represented approximately by the surface reduction (i.e. from the surface in between satenti and nsat; and the surface in between sat_{enti+1} and nsat). Thus we can easily prove that there is a resilient process over ev1 in between i and i+2 (L.e. $resil_{i,i+1}^{TF} \wedge resil_{i+1,i+2}^{TF}$ is true).

Table 6.13. Values for tolmax, tol, ltol.

Entity	$tolmax$	tol	$ltol$
ent_i	11	9,5	0.86
ent_{i+1}	11	7	0.64
ent_{i+2}	11	2.5	0.23

Table 6.14. Values for Δtol and Δfail.

$Entities\ couple$	Δtol	$\Delta fail$	$\Delta qfail$
ent_i, ent_{i+1}	0,23	1.5	2
ent_{i+1}, ent_{i+2}	0,41	1.5	1
ent_i, ent_{i+2}	0,64	0	3

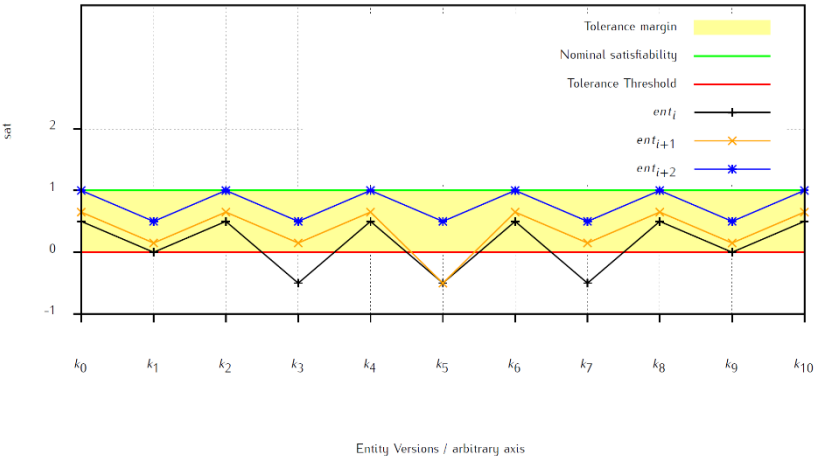


Figure 6.22. Simple Satisfiability correlation arbitrary axis / versioning axis

In the general case, nsat and tolsat may vary and sat_{enti+1} is not a constant improvement over the two evolution axes w.r.t. satenti. Figure 6.23 demonstrates such a case. From a surface point of view, we are interested in observing the tolerance surface (in between tolsat and nsat), which is between the satisfiability gap through the evolution axis (i.e. the surface in between sat_{enti} and sat_{enti+1}).

We can draw the following conclusions concerning Figure 6.23:

- (+) represents the tolerance that is removed by evolving from enti to ent_{i+1}. ent_{i+1} is thus considered as an improvement w.r.t. enti for the associated evolution steps over time.
- (-) is the tolerance that is added by evolving from enti to ent_{i+1}. It is a result of the fact that ent_{i+1} represents a degradation w.r.t. enti for the corresponding evolutions over time axis.
- (i) corresponds to failures not suppressed by the evolution.
- (ii) is a situation where the evolution has made a satisfactory entity evolve into a failing entity.
- (1) and (2) represent no improvement w.r.t. tolerance or to failure has been accomplished by the evolution. Both entity evolutions are over or below the tolerance margin.

If we compute Δfail and Δtol for the two entities for all the evolutions from k_0 to k_{10}, we notice that the tolerance level ltol decreases from 0.43 to 0.12 corresponding to Δtol of 0.21. The total level of tolerance is reduced by a factor 3.5. Thus we can deduce that as soon as Δtol is greater (resp. lower) than 0 we have an improvement (degradation) of tolerance needs due to the evolution. The number of failures, is constant at 3. Thus according to our definition there is no resiliency because we did not reduce the number of failures. Resilience can now be analysed by observing its values over the chosen evolution axis. The current notion of resilience is given in definition 45. It considers global tolerance and the total number of failures. We are free

to define specific notions of resilience depending on a number of interesting parameters. A non-restrictive list could be:

- minimum, maximum delays in term of evolution steps needed to change Δtol and Δqfail.
- use of Δqfail instead of Δfail.
- mandatory or optional removal of failures.
- focus on specific properties that might be different between evolution steps instead of fusing all properties to evaluate the degradation and the improvement.
- focus on specific evolution intervals at the dynamic evolution axis level in order to reduce the constraints on resiliency.
- provide constraints on the durability of improvement. The resilience from ent_i, to ent_j could be reguired to last over the next generations (no decadence).
-

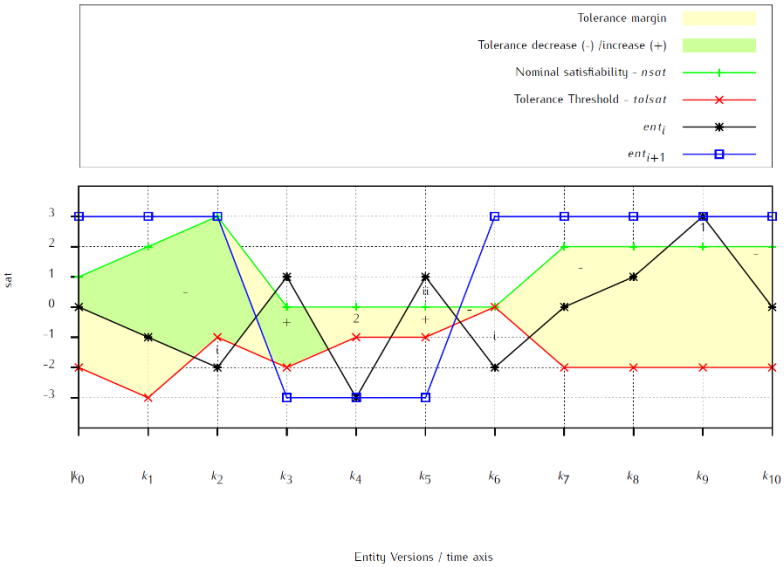


Figure 6.23 Advanced Satisfiability correlation arbitrary axis /versioning axis

Model Driven Engineering using the DREF framework

The objective is to allow the use of the DREF framework at the modelling level no matter what the modelling language is used. The motivation for this objective is clearly to allow modelling notation (DSL- Domain Specific modelling Language) providers to benefit from the DREF framework in order to include precisely defined concepts that address dependability and resilience. This will allow users of the modelling notation to explicitly and precisely address dependability concepts.

For this objective we propose an approach composed of the following steps:

- Define the meta-model of the targeted DSL using standard meta-modelling tools i.e. ecore equivalent diagrams [89]. The model should include meta concepts derived from the DREF meta-model.
- Define properties, entities and observers at the domain level.
- Define the evolution axis.
- Define the satisfiability functions including the nominal and the tolerance threshold.

We begin this section by presenting the DREF meta-model. Subsequently, each of these phases is illustrated using a concrete example of a toy DSL in the domain of business process modelling.

The DREF meta-model

In this section, let us provide a proposal for introducing DREF formal framework elements using meta-modelling techniques. The idea is to propose adequate meta-modelling elements to exploit the definition of a domain specific language that would be targeted by potential future users of DREF. In Figure 6.24 provide the model structure of the DREF meta-model is provided. The idea is that at the modelling level a DREF model to be composed of 3+1 categories of models is requested. The first three ones (modes view) are dedicated to the nominal view, the tolerance view, the fail view. The fourth provides the satisfiability view. First, it must be noticed that the modes view could allow one to provide several models per category. This should be defined in the specific meta-model structure. The models provided will propose a specific level of the separation of focus of these views. This means that for a specific DSL the modelling elements for each of these views might be strongly separated while in some other approaches they might be fused. It will be up to the meta-model designer to define the level of separation of concerns that is required.

In the following example, one can observe an approach for defining the level of separation of perspectives concerning a DREF extension of some BPMN like meta-model. The example provides a clear separation of modelling elements dedicated to nominal satisfiability, tolerance margin and failures (Figure 18). In any case, at the semantic level, these views must be implemented for having a means to determine if the satisfiability function should be:

- greater than or equal to the nominal satisfiability (Nominal mode);
- within the tolerance margin (Tolerance mode);
- or lower than the tolerance threshold (Fail mode).

Figure 8 represents a proposal for a meta-model fragment. First the meta-model here is incomplete. Second, the approach chosen to define the meta-model elements associated with the concepts defined in the formal definition of the DREF framework is very simple so as to be evidently clear. Concerning the incompleteness, many properties are not expressed and will need to be added. These properties should constrain any model satisfying this meta-model and should be compliant with the properties expressed in the formal definition. As an example, the meta-model requires the following properties: unique index values associated with an evolution axis; all tuples of a satisfiability function concerning the same evolution axis; and satisfiability of a property over an entity can only be defined if all observers have this entity and property in their definition domain. For this later property, we would have to parse all the tuples of the Dref_Satisfiability_tuple and verify that for all entity and property couples considered there exists a tuple for all the observers). Furthermore, if we want to impose domain-homogeneous satisfiability functions (cf. see definition 12 and example 20) we should constrain that there is no property for which there is at least one observer capable

of evaluating its satisfiability (satisfiability value set to a real number value) and at least one observer incapable of evaluating the satisfiability (satisfiability value set to the undetermined value ⊥).

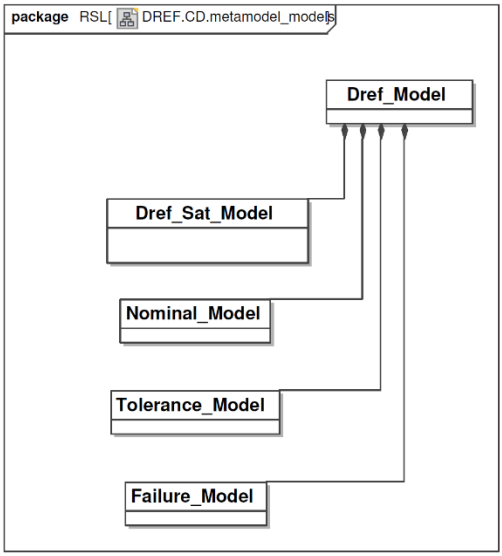


Figure 6.24. DREF Meta-model - Models view

The satisfiability view is a model of the satisfiability function. In Figure 6.25 an initial proposal for this concept of the DREF framework at meta-model level is given. Any Sat_model defines a satisfiability function. The meta-model elements given in Figure 6.25 are those that allow for modelling satisfiability functions in extension. This means that a compliant model would contain at least three Dref_Satisfiability functions: one for the nominal satisfiability, one for the tolerance threshold and at least one for the effective satisfiability for each evolution axis. Each Dref_Satisfiability function is defined by a set of

tuples roughly of the form: <evolution axis name and index, Entity, Observer, Observer Weight, Property, Property eight, a real value or the undetermined value>.

Of course, depending on the means available for defining such a function, it will be possible to add another model view in which the satisfiability function could be defined (statically or dynamically) differently. As an example, the function could be defined depending on an axiomatisation, the quantitative results of the evaluation of a test set (as it is done in validation testing), or as the quantitative results of the evaluation of a set of theorems (as done in model checking). In any case, the result will always be a satisfiability function that should comply with the meta-model proposed (i.e. the extensional or intentional views of these functions should be coherent).

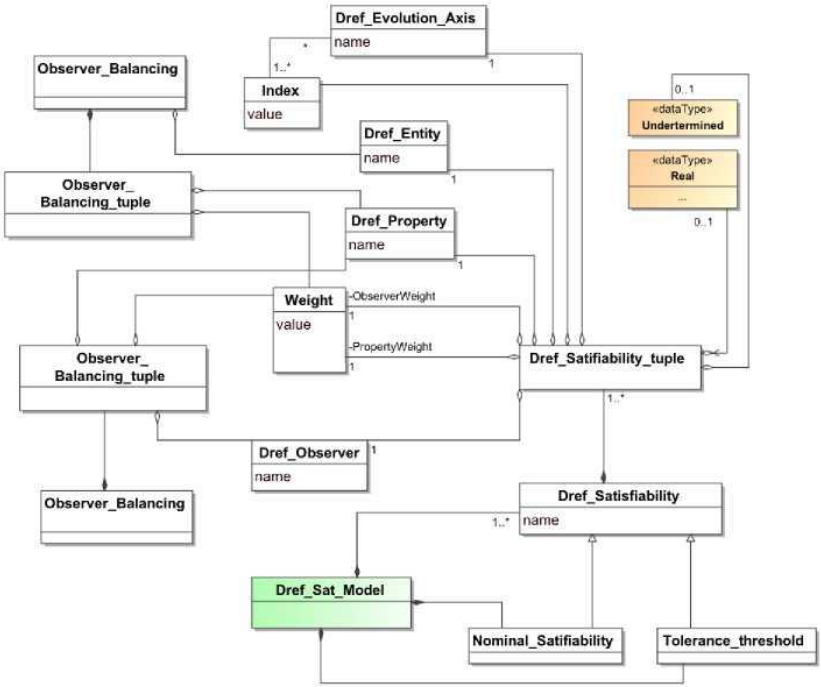


Figure 6.25. DREF Meta-model - Satisfiability view

Applying the DREF framework

In this section, a small experimental validation of the approach provided in this article, by engineering a small domain specific language, for modelling Resilient Business Processes is presented.

Illustration in the context of Business Process Modelling

In this section illustrates the approach in the context of a very simple version of a domain specific modelling language dedicated to the modelling of business processes. This DSL is built according to the BPMN (Business Process Modelling Notation) Standard [77].

The BPMN and DREF-BPMN Meta-models

In this illustration, let us choose to extend a BPMN like DSL with the concepts of the DREF framework. In order to proceed, we first have to define the meta-model of the BPMN-like DSL (provided in Figure 6.27). It indicates that a specification can contain three types of models: a BPMN model (example of an instance given in Figure 6.28), a constraint model (instance given in Figure 6.31) and two models subtypes of class diagram: concept model (instance given in Figure 6.29) and interface model (instance given in Figure 6.30). Furthermore, a BPMN model is a simplified standard BPMN [77] (i.e. pools, lanes, activities or tasks). The simple particularity is that we attach pre/post expressions using the Object Constraint Language (OCL) to tasks, thus providing an axiomatic specification of tasks (see Figure 6.31).

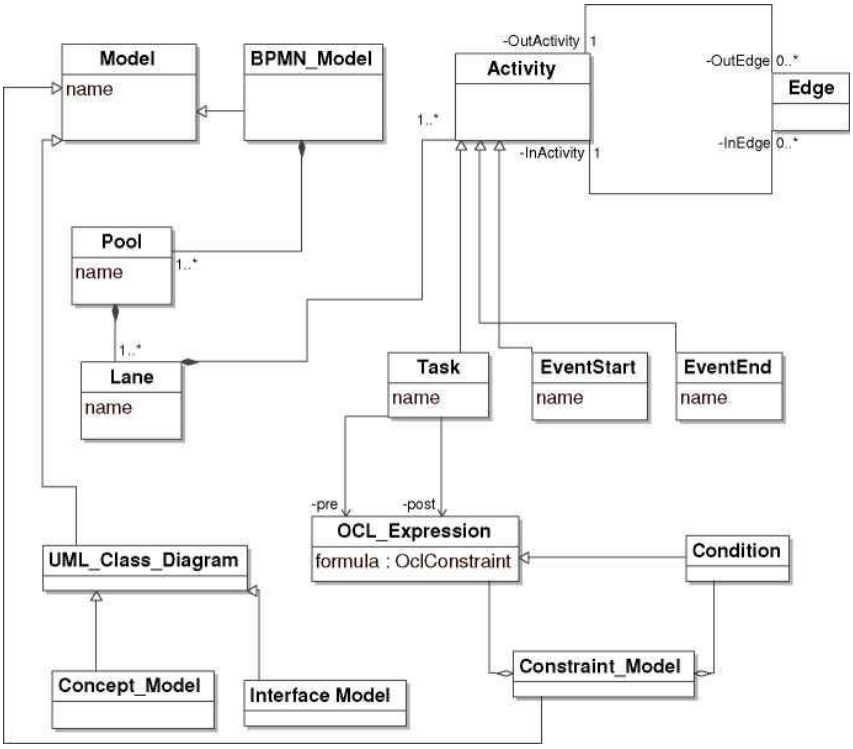


Figure 6.26. Simplified BPMN Meta-model

Let us define an extended BPMN meta-model demonstrating its integration with the DREF meta-model. Figure 10 illustrates such an extended meta-model. In the context of this simple example, we have defined a DREF model for which the nominal, tolerance and failure models are defined as BPMN models. Of course, the constraints provided for each of theses three models will define what is allowed to be included in these models14. It is important to note that the extension proposed concerns only the structural part which is related to the

models view of the DREF meta-model (see Figure 6.24). Concerning the satisfiability functions, the example given in this section does not provide the associated meta-models. The idea is nevertheless the same i.e. for each property and entity (in this example, business process instances) a satisfiability value is provided (see the graphical representation in Figure 16).

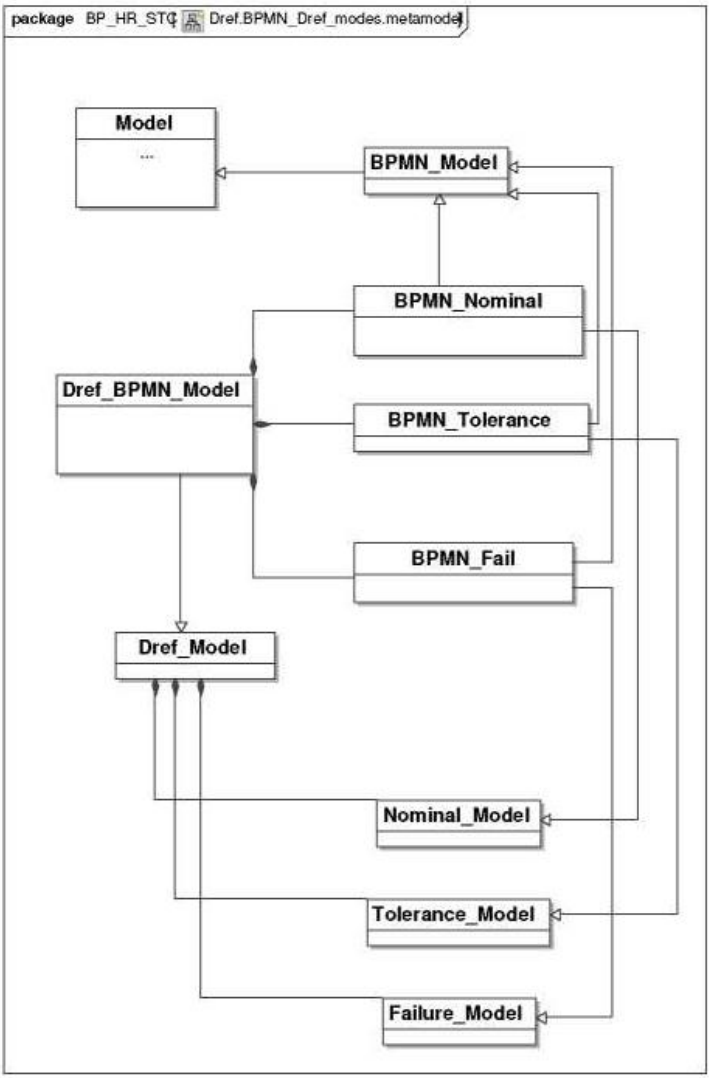


Figure 6.27. DREF-BPMN Meta-model

The Nominal Mode

Let us consider a simple Human Resources (HR) Department. The HR department has one business process which manages short term contracts (STC) for specific missions. Let us have the following informal requirements description: HR department has as a goal to process the STC recruitments for the different services of the company. Each time a service has a position to fill, he provides the position description to the HR such that HR can start the process. HR publishes the position using external diffusion (such as journals, web, national recruitment services). Applications are received until the position is closed i.e. when at least 2 applicants applied. All applicants are interviewed and then the HR selects the best candidate. The contract is signed with the selected candidate. Finally, company service and unselected applicants are notified.

According to the meta-model, this BPMN model for the nominal mode is made of a process model (cf. Figure 6.28), a concept model (cf. Figure 6.29 and 6.30) and a constraint model (cf. Figure 6.31). In Figure 6.28, the BPMN model of this STC contract BP of the HR department is we provided. This BP model should be considered as the process that the HR must apply in case of STC management.

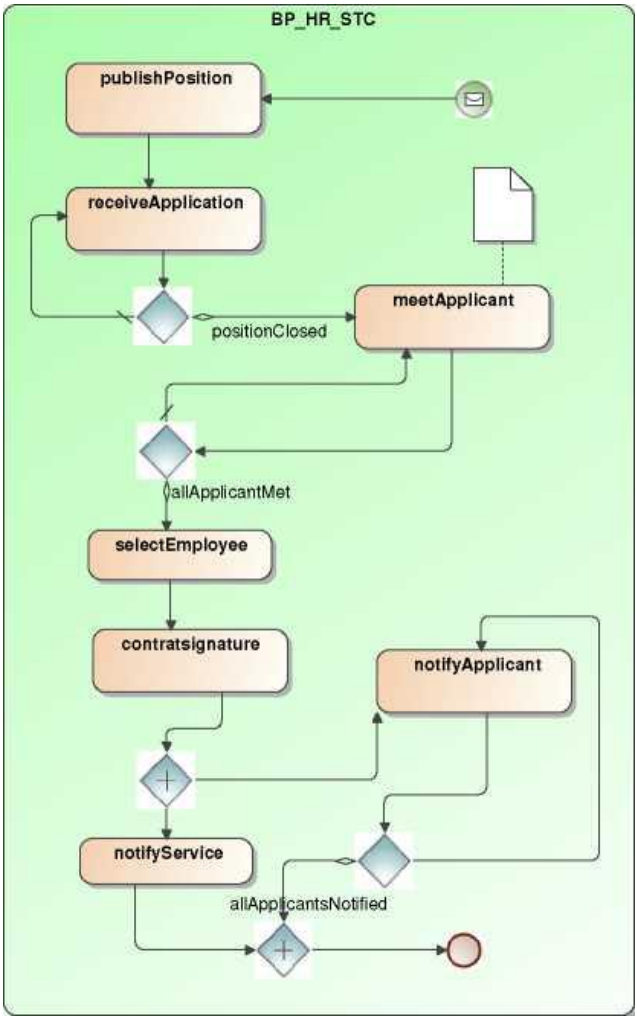


Figure 6.28. HR Short Term Contract BP - Nominal mode - process model

According to the meta-model (Figure 10), the activities are related to data which are modelled in the concept model (Figure 6.29). This data model more precisely describes the concepts, which are mentioned in the informal requirements (of course this should be performed in an iterative process involving all the concerned stakeholders).

In order to specify more precisely the activities of the BP let us provide for each of its activities (listed in the interface view of Figure 6.30) a pre/post axiomatic specification using OCL [90]. An example of such a specification is provided in Figure 6.31. In this specification, the pre-condition of the meetApplicant activity indicates that the HR department has not previously interviewed the candidate (if this is not the case, the outcome of this activity is not known and the fault will not be attributed to the HR). The post-condition, indicates that the HR registers the fact that this applicant has been interviewed.

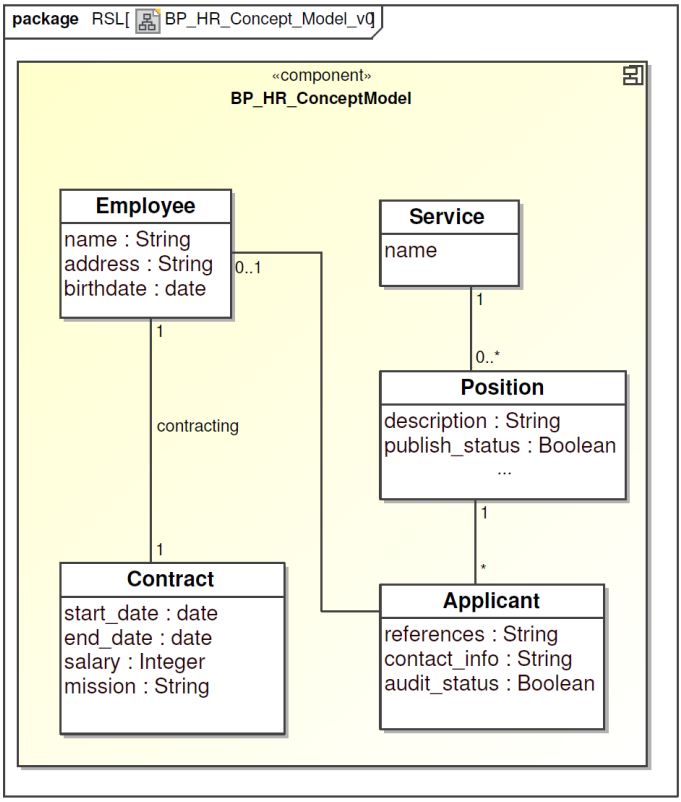


Figure 6.29. HR Short Term Contract BP - Nominal mode - Concept model

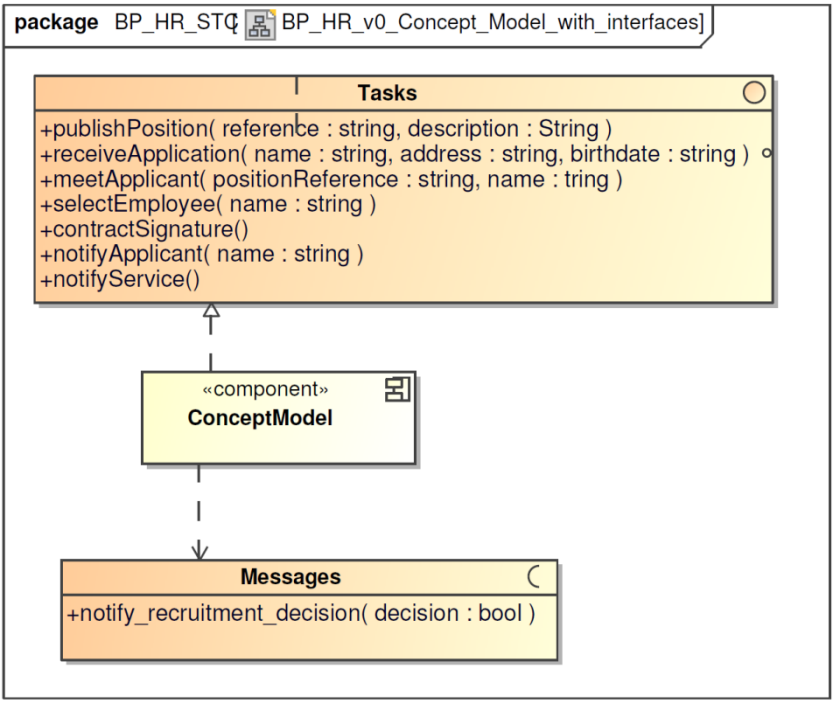


Figure 6.30. HR Short Term Contract BP - Nominal mode - Process Activities and Messages

Pre- and post-conditions are part of the constraints' model as well as theconditions used in the BP model for gateways. In the same model, the positionClosed logical property is specified as a Boolean operation that is true iff the total number of applicants met is more than one.

Entities, Observers, Properties and Balancing

Having defined the nominal view of the business process let us now define Entities, Observers, Properties and Balancing in the following way:

```
context BP_STC::Tasks::meetApplicant(positionReference:String,name:String):OclVoid
pre:
let theApplicant:BP_STC::ConceptModel::Applicant in
theApplicant.name = name
and theApplicant.audit_status=false
post:
let theApplicant:BP_STC::ConceptModel::Applicant in
theApplicant.name = name
and theApplicant.audit_status=true
```

```
context BP_STC::ConceptModel
inv positionClosed :
let allApplicantsMet:Set(BP_STC::ConceptModel::Applicant) in
allApplicantsMet = (BP_STC::ConceptModel::Applicant).allInstances()
                   ->collect(audit_status=true)
and allApplicantsMet->size()>1
```

Figure 6.31. HR Short Term Contract BP - Nominal mode - constraints model extract

- Entities: let us propose to consider real business processes executions as entitles . A real business process Is given In terms of a description of what has been executed by the HR department. In order to continue In a consistent way with the model driven engineering approach, a real business process will be given as a model. In the case of seguentlal semantics where the participants In the BP are not distinguishable, they should correspond to a set of directed graphs representing the seguence of activities and messages labelling links between nodes representing the BP state . To simplify, we will just model them as a an ordered set of events being activity or message Identifiers. As an example, we consider the four BP instances modelled as oriented graphs in Figure 6.32.
- Observers: Each of these BP instances are observed by only one observer who is the Quality Control Officer (QCO) of the company.
- Properties: The QCO is interested in observing the entities using the model that has been given to the HR department. In this very

simple case, we consider that the property is provided by the BP Process Models' view (the conjunction of the models provided in Figures 6.28-6.31).

- Balancing: In this very simple illustration we have one observer and one property. We thus have a default balancing such that the weight of the observer and the property are egual to 1.

Evolution and Observation axis

Let us define two axes for the purpose of illustrating the DREF framework in model driven engineering and development processes:

- An evolution axis (ev) has two indexes (1 and 2) representing one evolution step of the HR service. While in the first one the HR employee activities would be evaluated according to a non fault-tolerant business process (modelled in Figure 6.43). The second version of the HR department introduces a tolerance margin (described in Figure 6.34).
- An observation axis (obs) with four indexes (1 to 4) to denotes the 4 observation points is shown in Figure 6.32. They represent four completed recruitment processes executed by the actual HR Service.
-

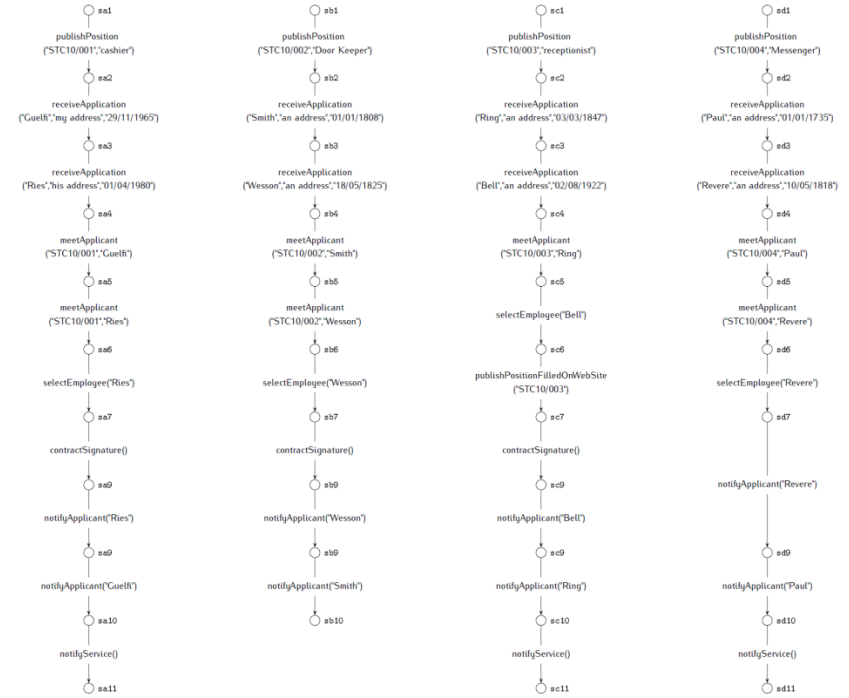


Figure 6.32. Actual BP instances observed at HR Department

The Satisfiability view

Once the entitles distributed along the given evolution axes, properties and observers (with or without balancing) are defined, we have to provide the satisfiability view as a model Instance of the meta-model. In the case of the toy example, we have one observer (HRD as the director of the HR service), one property as the business process model given In Figure 6.28 and eight entitles ($bpi_0^1, \dots, bpi_3^2$).

Satisfiability view Is then given as a set of tuples defining the satisfiability functions. In the context of the case study, we represent them by the two plots given in Figure 6.33.

In the Illustrations, the values of the satisfiability functions are determined using the nominal and tolerance modes provided In the corresponding models given in Figure 6.34. The satisfiability function Is such that:

$sat(bpi_i^j) = 1\ iff\ bpi_i^j$ belongs to the business process Instances that are "Instances" of the Nominal model (i.e. compiles with the Nominal mode definition).

$sat(bpi_i^j) = 0.5\ iff\ bpi_i^j$ belongs to the business process instances that are ”instances” of the Tolerance model (i.e. it complies with the Tolerance mode definition).

$sat(bpi_i^j) = 0\ iff\ bpi_i^j$ is any other business process instance.

-
-

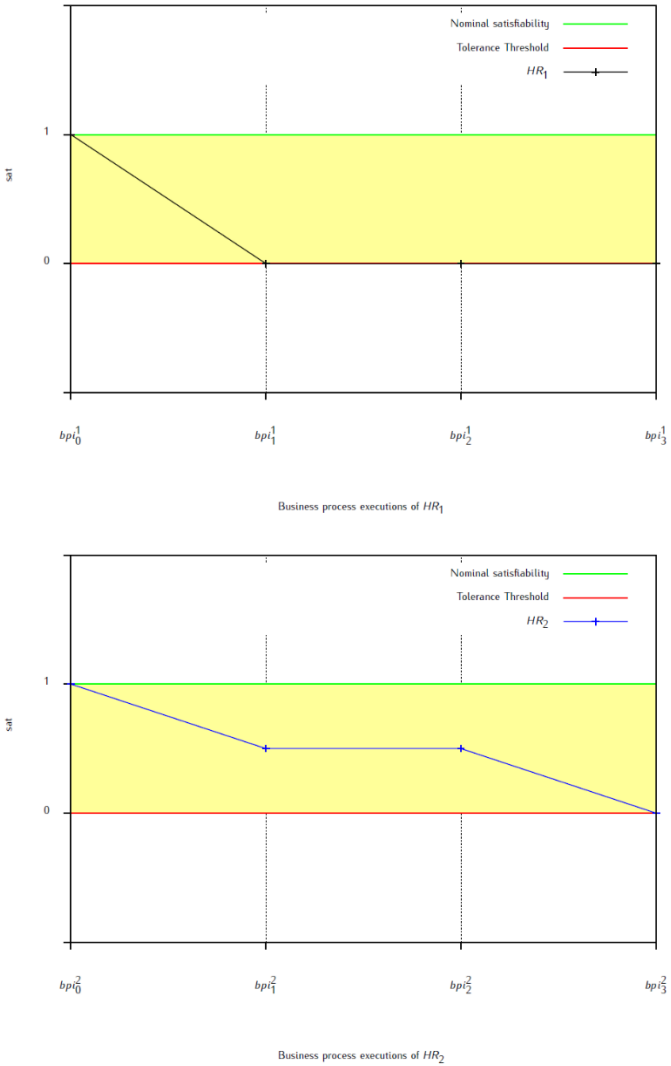


Figure 6.33. Satisfiability for HR version 1 & 2

In Figure 6.5 the Failure view(s), which is used here only to model a subset of the business process instances for which the satisfiability function returns a failure (below the tolerance margin), is included. Depending on the DREF extension targeted, either this view could be considered as fully characterising the failures or only a subset. In the first case, any business process instance not in the nominal, tolerance, or failure sets should either not be in the definition domain of sat or the sat value should be $\perp$. In the second case, the view should be used to document a subset of failing business process instances.

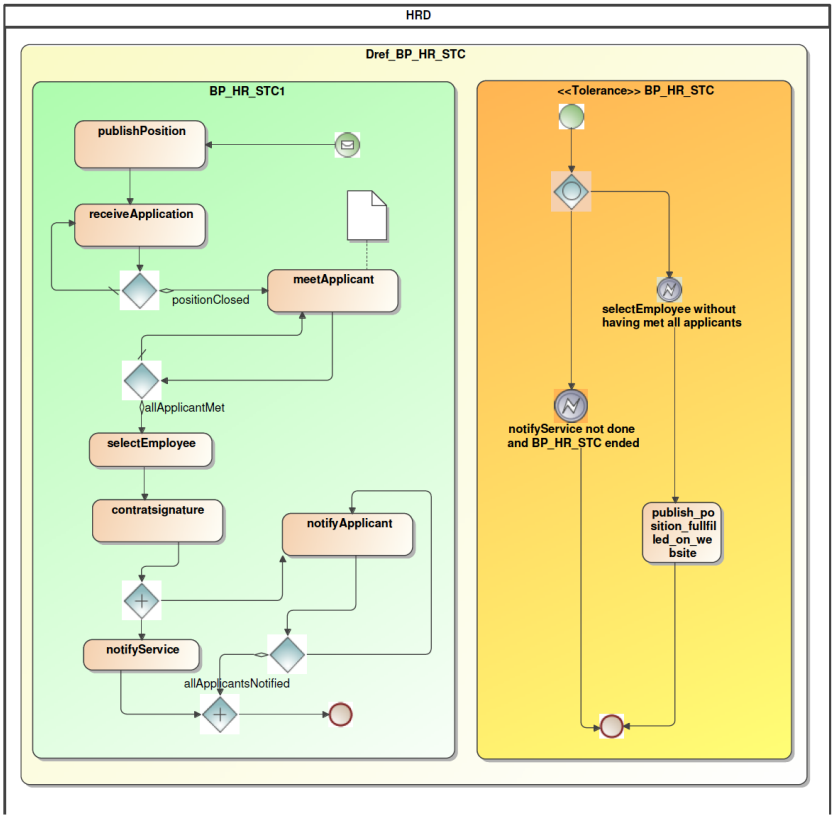


Figure 6.34. HR Short Term Contract BP - process model with tolerance modelling

In the case of failure modes, the view could be used to partition the failure margin in failure modes (the red variants). This can be generalised to the tolerance view, which would provide a means to model the degradation modes.

Tolerance, failure and Resilience

Given the satisfiability functions, we can now analyse the basic dependability elements, which are failure and tolerance, and the T −Resilience;F −Resilience; TF −Resilience over the evolution axes. In our case study, we have $\Delta q\ Fail_{1,2} = 3 - 1 = 2\ and\ \Delta tol_{1,2} = 13 - 12 = 1$. Thus we can say that there is T-Resilience and F-Resilience (and so TF-resilience) in the evolution process from HR_1 to HR_2.

Other Approaches for DREF Satisfiability functions

In the context of safety critical system development, let us addres the problem of defining a development process that would improve the dependability of engineered software along the versioning evolution axis. The context for this problem was the development of embedded safety critical software for a car's airbag opening system. The approach was based on a validation using testing [91]. Thus each software version was validated based on the results of a set of test cases. A constraint was to ensure non-regression in the versioning. The meta-models for the nominal view were provided in terms of protocol state machine and class diagrams with OCL constraints. After a thorough analysis of the application domain, it has been concluded that the validation would be dependent on 5+2 views (cf. Figure 6.36). Each of these views would then select a set of test cases to be used to define the satisfiability value of the software version. Even if it would be appropriate, the first approach used did not define a balancing, neither between the views nor between the test cases.

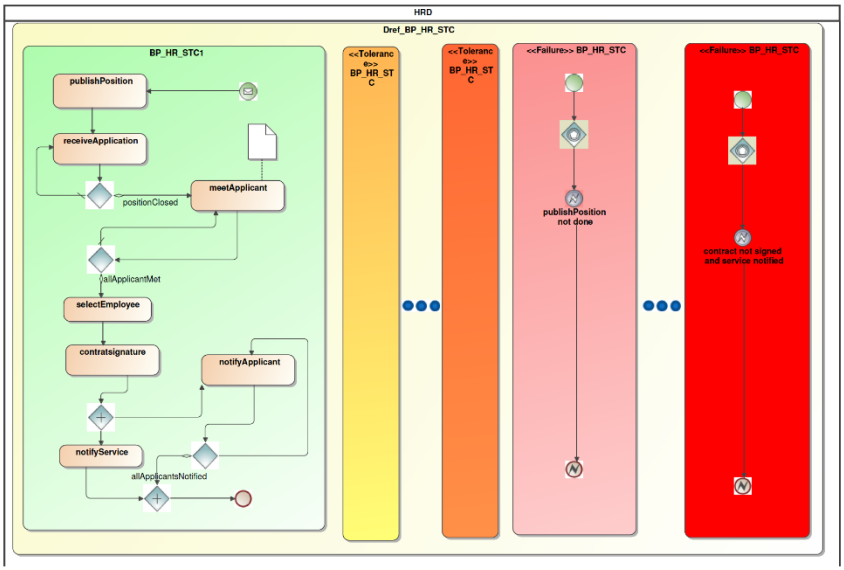


Figure 6.35. HR Short Term Contract BP - process model with tolerance and Failure modelling

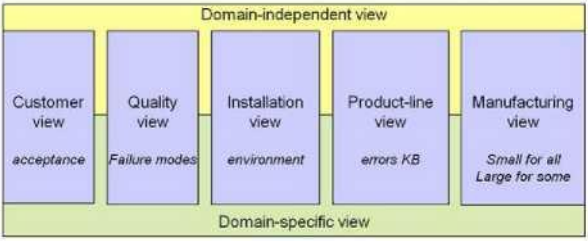


Figure 6.36. Views

Using test results as a mean to compute the satisfiability is a very valuable and pragmatic approach for DREF. From a similar perspective, model checking techniques could also be considered.

Practical use of the DREF framework is currently limited to the field of DSL development (as illustrated by the case study). Currently additional experiments to better assess the framework are conducted. One experiment in the field of operational resilience [92] uses entities as algebraic Petri net models [93], properties as invariants regarding places in the APN, and satisfiability as logical functions computed using the model checker AlPiNA [94]. A second experiment was performed in the context of architecture description languages [95]. The aim was to improve the AADL modeling language with the DREF concepts, thus engineering an architecture description language for resilient architectures. Those experiments provide a practical assessment of the usefulness of the approach presented in this article for the specific aims targeted.

Conclusion

In this section, an initial version of a formal framework DREF that precisely defines the fundamentals concepts used to define dependability and resilience of ICT systems is introduced. This framework has been defined using set theory at a chosen abstraction level in order to cover the current advances in the terminology of dependable and resilient systems engineering. The proposed framework has been designed to be useful for ICT system model driven engineering. To this extent, a meta-model has been proposed for the framework and a validation of the approach using the DREF framework for modelling language engineering has been provided.

Advancement questions

1. What is the KAOS?
2. What does KAOS specification language provide?
3. What we should do to formally express these security goals?
4. What we need to do for applying the generic security model to a system?

5. What is the difference between the OR and AND-refinement?
6. What is the FADSE framework?
7. What are the main limitations of the semi-formal approaches to security engineering?
8. What is the difference between the dependability and resilience?
9. What is the model driven engineering?
10. What is main purpose of the DREF?

REFERENCES

1. Hassan R. Formal Analysis and Design for Engineering Security / R. Hassan // PhD thesis. – Blacksburg, Virginia. - 2009. - p. 235.
2. A. van Lamsweerde Handling Obstacles in Goal-Oriented Requirements Engineering / A. van Lamsweerde, E. Letier // IEEE Transactions on Software Engineering, Special Issue on Exception Handling. – 2000. - Vol. 26, No. 10. – pp. 978-1005.
3. Lamsweerde, A. Building Formal Requirements Models for Reliable Software / A.Lamsweerde // Proceedings of the 6th Ade- Europe International Conference Leuven on Reliable Software Technologies.-2001.- pg. 1-20.
4. A. van Lamsweerde Elaborating Security Requirements by Construction of Intentional Anti-Models / A.Lamsweerde// Proc. ICSE'04: 26th Intl. Conf. on Software Engineering.-2004.
5. Landtsheeer, R. Reasoning about Confidentiality at Requirements Engineering Time / R.Landtsheeer, A.Lamsweerde // Proceedings of the 19h European Software Engineering Conference held jointly with 13th ACM SIGSOFT

International Symposium on Foundations of Software Engineering. - 2005.- pg. 41-49.
6. Ponsard, C. Early Verification and Validation of Mission Critical Systems / C Ponsard,., et. al. // Proceedings of the 9th International Workshop on Formal Methods for Industrial Critical systems (FMICS).- 2004.
7. Letier, E. Agent-Based Tactics for Goal-Oriented Requirements Elaboration / E.Letier, A.Lamsweerde, // Proceedings of the 24th International Conference on Software Engineering.- 2002.- pg. 83-93,.
8. Letier, E. Deriving Operational Software Specifications from System Goals / E.Letier, , A.Lamsweerde, // ACM SIGSOFT Software Engineering Notes.-Vol. 27, Issue 6.-pg. 119-128, 2002.
9. Fontaine, P.J. Goal-Oriented Elaboration of Security Requirements, M.S. Thesis / P.J Fontaine // Dept. Computing Science : University of Louvain, 2001.
10. Lamsweerde, A. Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice / Lamsweerde, A. // Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04).- 2004.
11. Darimont, R. Formal Refinement Patterns for Goal-Driven Requirements Elaboration / R.Darimont, A. van Lamsweerde // Proc. Fourth ACM SIGSOFT Symp. Foundations of Software Eng..- 1996.-pp. 179-190.
12. Letier, E. Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering / E.Letier, A.Lamsweerde, // ACM SIGSOFT Software Engineering Notes.-2004.- Vol. 29, Issue 6.- pg. 53-62.

13. Lamsweerde, A. Leaving Inconsistency / A.Lamsweerde, et. al. // Proceedings of the ICSE'97 Workshop on "Living with Inconsistency".-1997.
14. van Lamsweerde A. Goal- Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned / A. van Lamsweerde, R.Darimont, P.Massonet // Proc. Second Int'l Symp. Requirements Eng. (RE '95).- 1995.
15. Landtsheer, R. Reasoning about Confidentiality at Requirements Engineering Time / R.Landtsheer, A.Van Lamsweerde, // Proceedings of the 10th European software Engineering Conference.-2005.- pg. 41-49.
16. Letier, E. High Assurance Requires Goal Orientation / E.Letier, A. Van Lamsweerde // International Workshop Requirements for High Assurance Systems.- 2004.
17. Fontaine, P.J., Goal-Oriented Elaboration of Security Requirements, M.S. Thesis/ P.J.Fontaine, // Dept. Computing Science: University of Louvain.-2001.
18. J. Viega Building Secure Software: How to Avoid Security Problems the Right Way / J. Viega, G. McGraw // Addison-Wesley.- 2001.
19. Wing A Symbiotic Relationship Between Formal Methods and Security / Wing, Jeannette // Proceedings of Computer Security, Dependability and Assurance: From Needs to solutions.-1998.- pg. 26-38.
20. Beeck A. Formal Requirements Engineering Method for Specification / Beeck, V.Michael, Margaria, Tiziana, Steffen, Bernhard, // Synthesis, and Verification, in Proceedings of the 8th conference on Software Engineering Environments.-1997.- pg. 131-144.

21. B. Nuseibeh A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications / B. Nuseibeh, J. Kramer, A. Finkelstein // IEEE Transactions on Software Engineering.-1994.-Vol. 20 No. 10.-pg.760-773.
22. Van Lamsweerde, A. Formal Specification: a Roadmap / A.Van Lamsweerde, A. Finkelstein // Proceedings of the Future of Software Engineering.- ACM Press, 2000.
23. J. Mylopoulos From Object-Oriented to Goal-Oriented Requirements Analysis / J. Mylopoulos, L. Chung, E. Yu // Communications of the ACM.-1999.- №42(1):31-37.
24. Y.Yu From stakeholder goals to high-variability software design. Tech. Rep. CSRG-509 / Y.Yu, et. al..-Canada,Ontario: University of Toronto, 2005.
25. William Heaven A UML profile to support requirements engineering with KAOS / William Heaven and Anthony Finkelstein // IEE Proceedings: Software.-2004.- Vol.151, Issue No. 1, pg. 10-27.
26. L. Jiang Incorporating Goal Analysis in Database Design: A Case Study from Biological Data Management. / L. Jiang, T. Topaloglou, A. Borgida, J. Mylopoulos // RE'.-2006.-pg. 196-204.
27. Bowen, J.Application of Formal Methods / J.Bowen, M.Hinchey // Prentice Hall.- 1995
28. Clarke Formal Methods: State of the Art and Future Directions / Clarke, Edmund, J.Wing, et. al. // ACM Computing Surveys.-1996.- Vol. 28, No. 4.
29. The Common Criteria for Information Technology Security Evaluation (CC)[Electronic resourse].- Access mode:http://www.commoncriteriaportal.org/

30. Stepney An Electronic Purse Specification, Refinement, and Proof / Stepney, Susan, Cooper, et. al. // Programming Research Group: Oxford University Computing Labarotory.-2000.
31. Sabatier, D. The Use of the B Formal Method for the Design and the Validation of the Transaction Mechanism for Smart Card Applications / D.Sabatier, L.Pierre// In the 17th Proceedings of Formal Methods in System Design.-2000.- pg. 245-¬272.
32. McDermott J. Using abuse case models for security requirements analysis / J.McDermott, C.Fox // Proceedings of the 15th annual computer security applications conference (ACSAC'99),Phoenix, Arizona.-1999.-№1.
33. G. Sindre Eliciting Security Requirements by Misuse Cases / G. Sindre, A.L. Opdahl // Proc. TOOLS Pacific'2000 - Techn. of Object-Oriented Languages and Systems.-2000.- pg.120-131.
34. Firesmith Security Use Cases[Electronic resourse] / Firesmith, G.Donald // Journal of Object Technology (JOT).-2003.-issue 5.- Access mode:http://www.jot.fm/issues/issue 2003 05/column6
35. L. Liu Security and Privacy Requirements Analysis withina SocialSetting / L. Liu, E. Yu, J. Mylopoulos // Proc.RE'03-Intl. Conf. Requirements Engineering.- 2003.-pg.151-161.
36. L. Chung Nonfunctional requirements in software engineering. / L. Chung, B. Nixon, E. Yu, J. Mylopoulos.- Boston:Kluwer Academic,2000.
37. Yu, E. Why Agent-Oriented Requirements Engineering / E.Yu // Proc. of the 3rd Int. Workshop on Requirements Engineering: Foundations for Software Quality, Barcelona, Catalonia. E.

Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur.-1997.
38. Nakagawa, H. Formal Specification Generator for KAOS / H.Nakagawa, K.Taguchi, S.Honiden // Proceedings of The International Conference on Automated Software Engineering (ASE'07).-2007.
39. Brandozzi, M. Transforming Goal Oriented Requirement Specifications into Architectural Prescriptions / M.BRANDOZZI, D. E. PERRY.- Toronto, Canada:First International Workshop from Software Requirements to Architectures (STRAW'01).
40. Van Lamsweerde, A. From System Goals to Software Architecture. / A.Van Lamsweerde // in Formal Methods for Software Architectures. M. Bernardo and P. Inverardi, Springer-Verlag.-2003.-pg.25-43.
41. Liu, L. From Requirements to Architectural Design - Using Goals and Scenarios./L.Liu, E.Yu // ICSE- 2001 Workshop: From Software Requirements to Architectures (STRAW2001).-2001.- pp. 22-30.
42. J. Mylopoulos UML for agent-oriented software development:The Tropos proposal / J. Mylopoulos, M. Kolp, J. Castro // Proc. of the 4th Int. Conf. on the Unified Modeling Language UML.-2001
43. R. G. Dromey From requirements to design: Formalizing the key steps / R. G. Dromey // International Conference on Software Engineering and Formal Methods: The University of Queensland School of Information Technology and Electrical Engineering,2003.- pp. 2.13.

44. M.G. Hinchey Requirements to Design to Code: Towards a Fully Formal Approach to Automatic Code Generation / J.L. Rash, C.A. Rouff // NASA Technical Report.-2005.
45. Hoss Ontological Approach to Improving Design Quality / Hoss, M.Allyson, Carver, L.Doris // in Proceeding of IEEE Aerospace Conference.-2006.- pg.12.
46. J. Liu Linking UML models of design and requirement / J. Liu, Z. Liu, J. He, X. Li.-Melbourne, Australia:IEEE Computer Society. 82, 2004.
47. X. Li Formal and use-case driven requirement analysis in UML / X. Li, Z. Liu, J. He.-Illinois, USA: IEEE Computer Society,2001
48. Z. Liu A relational model for formal object-oriented requirement analysis in UML / J. He, X. Li, Y. Chen // Lecture Notes in Computer Science 2885. Springer.- 2003.
49. Hung Ledang Formalizing UML behavioral diagrams with B. In Tenth OOPSLA Workshop on Behavioral Semantics : Back to Basics / Hung Ledang and Jeanine Souquieres.- Tampa Bay, Florida, USA, 2001.
50. Hung Ledang MODELING CLASS OPERATIONS IN B: application to UML behavioral diagrams. / Hung Ledang, Jeanine Souquieres.- Loews Coronado Bay, San Diego, USA,: IEEE Computer Society,2001.
51. Hung Ledang Integration of UML Views Using B Notation / Hung Ledang, Jeanine Souquieres // In proceedings of WITUML02,Spain.-2002.
52. Blackburn, M. Removing Requirements Defects and Automating Test / M.Blackburn, R.Busser, A.Nauman, // Software Productivity Consortium.-2001.

53. Busser, R. Automated Model Analysis and Test Generation for Flight Guidance Mode Logic / R.Busser, M.Blackburn, A. Nauman // The 20th Conference
54. Blackburn, M. Why Model-Based Test Automation is Different and What You Should Know to Get Started / M.Blackburn, R.Busser, A.Nauman // Software Productivity Consortiu.-2004.
55. Busser, R. Reducing Cost of High Integrity Systems through Model-Based Testing / R.Busser, et. al. // 23rd Conference on Digital Avionics Systems.-2004.-Vol. 2.- pg.6.B.1-61-13.
56. Massacci Using a Security Requirements Engineering Methodology in Practice: The Compliance with The Italian Data Protection Legislation / Massacci, Fabio, et.al.// Computer Standards & Interfaces.-2005.-issue 27.- pg. 445-¬455.
57. Van Lamsweerde, A. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering / A.Van Lamsweerde, E.Letier // Proceeding of Radical Innovations of Software and Systems Engineering, LNCS.-2004.-pg. 325-340.
58. K. Yue What Does It Mean to Say that a Specification is Complete? / K.Yue// Proc. IWSSD- 4, FourthInternational Workshop on Software Specification and Design,.- 1987.
59. A. van Lamsweerde Requirements Engineering in the Year 00: A Research Perspective / A.van Lamsweerde.- ACM Press:Invited KeynotePaper,2000.-pg.5-19.
60. A. Dardenne Goal-Directed Requirements Acquisition / A.Dardenne, A. van Lamsweerde, S. Fickas // Science of Computer Programming.-1993.- Vol. 20.- pg.3-50.

61. Objectiver : The power tool to engineer your technical and business requirements [Electronic resource] Access mode : www.objectiver.com
62. Van Lamsweerde, A. Managing Conflicts in Goal-driven Requirements Engineering / A.Van Lamsweerde, R.Darimont, E.Letier // IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management in Software Development.-1998.- Vol. 24, No. 11.- pg. 908-926.
63. Hassan Goal-Oriented, B-Based, Formal Derivation of Security Design Specifications from Security Requirements / Hassan, Riham, et. al. // Symposium on Requirements Engineering for Information Security, Barcelona, Spain.-2008.
64. Hassan Integrating Formal Analysis and Design to Preserve Security Properties / Hassan, Riham, et. al. // Proceedings of the HAWAII International Conference on System Science.-2009.
65. Cimitile, A. A Software Model for Impact Analysis: A Validation Experiment / A.Cimitile, A.R.Fasolino, G.Visaggio // Proceedings of the Sixth Working Conference on Reverse Engineering.-1999.- pg. 212-222.
66. The industrial tool to efficiently deploy the b method [Electronic resource] Access mode : http://www. atelierb.eu/index en.html.
67. B-core [Electronic resource] Access mode : http://www.b-core.com.
68. Rashid Bin Muhhamed Depth-First Search [Electronic resource] Access mode : http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/depthSearch.htm
69. http://www.cs.auckland.ac.nz/~ute/220ft/graphalg/node10.html.

70. Clarke, L. A Formal Evaluation of Data Flow Path Selection Criteria / L.Clarke, A.Podgurski, D. Richardson // IEEE Transactions on Software Engineering.-1989.-Vol. 15, No. 11.
71. Nicolas Guelfi A formal framework for dependability and resilience from a software engineering perspective / Nicolas Guelfi // Central European Journal of Computer Science. – 2011. – 294. – pp. 294-328.
72. Naur P. Software engineering report of a conference sponsored by the NATO science committee Garmisch Germany 7th-11th October 1968 / Naur P., Randell B. – Access mode : http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.
73. Randell B. Software engineering, As it was in 1968 / Randell B. // ICSE. – 1979. – p. 1-10.
74. Atkinson C. Model-driven development [Electronic resource] / Atkinson C., Kuhne T. // A metamodeling foundation IEEE Software. - 2003. – vol. 20(5). - p. 36-41, - Access mode : https://csdl.computer.org/comp/mags/so/2003/05/s5036abs.htm.
75. Ludewig J. Models in software engineering. / Ludewig J. // Software and System Modeling. - 2003. - vol. 2(1). - p. 5-14
76. OMG, Uml 2.0 infrastructure specification. Tech. rep., Object Management Group, 2003
77. OMG, Business Process Modeling Notation (BPMN), Version 1.2. - 2009. – Access mode : http://www.omg.org/spec/BPMN/1.2/PDF.
78. Harel D. Statecharts. A visual formalism for complex systems / Harel D. // Science of Computer Programming. - 1987. – vol. 8(3). - p. 231-274.

79. Avizienis A. Fundamental concepts of dependability / Avizienis A., Laprie J.C., Randell B. // Tech. rep., Computer Science Department, University of California, Los Angeles, USA. – 2001.
80. Avizienis A. Basic Concepts and Taxonomy of Dependable and Secure Computing / Avizienis A., Laprie J.C., Randell B., Landwehr C.E. // IEEE Trans. Dependable Sec. Comput. - 2004. – vol. 1(1). - p. 11-33.
81. Dearnley P.A. An investigation into database resilience / Dearnley P.A. // Comput. J. - 1976. – vol. 19(2). - p. 117-121.
82. Black P.E. Verifying resilient software / Black P.E., Windley P.J. // -1997. - p. 262-266.
83. Mostert D.N.J. A technique to include computer security, safety, and resilience requirements as part of the requirements specification / Mostert D.N.J., von Solms S.H. // J. Syst. Softw. - 1995, - vol. 31(1). - p. 45-53.
84. Svobodova L. Resilient distributed computing / Svobodova L. // IEEE Trans. Software Eng. – 1984. - vol. 10(3). – p. 257-268
85. Wirsing M. Algebraic specification / Wirsing M. // Handbook of Theoretical Computer Science. – 1990. - p. 675-788.
86. Ledyayev R. Architectural framework for product line development of dependable crisis management systems / Ledyayev R. // PhD thesis, Universite Henri Poincare Nancy. – 2009.
87. Erl T. Service-Oriented Architecture, Concepts, Technology, and Design / Erl T. - Prentice Hall. – 2006.
88. Pohl K. Software Product Line Engineering: Foundations, Principles and Techniques, 1 edn. / Pohl K., Böckle G., van der Linden F.J. - Springer, 2005.

89. Budinsky F. Eclipse Modeling Framework / Budinsky F., Brodsky S.A., Merks E. - Pearson Education. – 2003.
90. OMG, Ocl 2.0 specification. Tech. rep., Object Management Group. - 2005
91. Guelfi N. Selection, evaluation and generation of test cases in an industrial setting, a process and a tool / Guelfi N., Ries B. // Testing, Academic & Industrial Conference, 47-51, IEEE, Windsor, UK. – 2008.
92. Lúcio L. A precise definition of operational resilience / Lúcio L., Guelfi N. // Tech. Rep. TR-LASSY-11-02, Laboratory for Advanced Software Systems, University of Luxembourg. – 2011.
93. Reisig W. Petri nets and algebraic specifications / Reisig W. // Theor. Comput. Sci. – 1991. - p. 1-34.
94. A symbolic model checker / Buchs D., Hostettler S., Marechal A., Risoldi M., Alpina // Lecture Notes in Computer Science. Springer. - 2010. - Vol. 6128.
95. Saidane A. Dref resiliency and security aspects in the sae architecture analysis and design language (aadl) / Saidane A. // Tech. Rep. TR-LASSY-10-07, University of Luxembourg. - 2010.

7 FORMAL METHODS FOR ARCHITECTING SECURE SOFTWARE SYSTEMS

Content of the CHAPTER 7

Introduction

Systematical engineering security into software applications is an important and difficult problem [1, 2, 3, 4] The importance of the problem can be seen from the number of security incidents reported to the Computer Emergency Readiness Team Coordination Center (CERT/CC) and their associated costs.The CERT/CC data from 2003 reports 137,529 incidents; the cost of electronic crimes is reported at 666 million dollars [5]. Most of these incidents, which can involve from one to thousands of sites, result from software vulnerabilities. The CERT/CC data indicate the number of these incidents continues to rise. The difficulty of the problem stems from its breadth, as it covers many areas such as authentication, auditing, authorization, confidentiality, integrity, and non-repudiation (security standard ISO 7498-2) [6], where authentication verifies the claimed identity of a user or provider, auditing ensures that user activity is properly recorded and reviewed, authorization moderates information use and provision, confidentiality means information is provided only for proper use as appropriate to the sensitivity of the information, integrity makes certain that information is used in ways that allow only necessary changes, and non-repudiation ensures the identity of a user is irrefutably verified and recorded as protection against their later denying participation. Each of these can be further categorized. For example, authentication may include peer entity authentication and data origin authentication; confidentiality may include connection confidentiality, connectionless confidentiality, selective field confidentiality, and traffic flow confidentiality, etc., as presented in Figure 7.1.

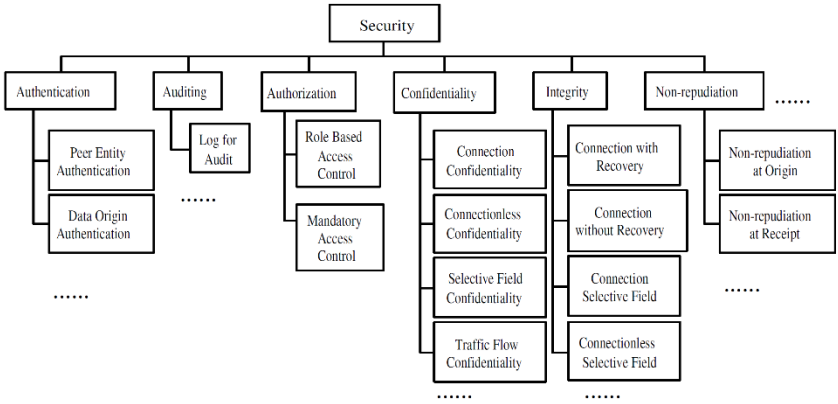


Figure 7.1. Elements of security

Security needs to be considered during each phase of the software development life-cycle, including requirements specification and analysis, architecture design, detailed design, implementation, testing, and deployment. The software architecture of a system is the structure of the system which comprises software elements, these elements' externally visible properties, where externally visible properties refer to their provided services, performance characteristics, fault handling, shared resource usage, and so on, and the relationships among those elements [7]. Software architecture focuses on designing and specifying the overall system's gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives etc.

As the first design phase, it is widely recognized that decisions made at the architecture design stage have a strong impact on the quality of the final product [8]. Hence, to provide a positive impact, architecture designs, which reflect architectural decisions, should be analyzed so that design flaws can be detected and removed. Discovering and fixing defects at the architecture design stage is more

cost- and time-effective compared to performing such work after the system is implemented, as fixing defects at the implementation stage would necessarily cause the revision and reconstruction of numerous design, implementation, and testing artifacts. Therefore, the architectural design and analysis of security properties is a very important step in the software development lifecycle. The architectural design of security properties enables the realization of a system's security non-functional requirements; the analysis of security properties provides architects with objective results to evaluate design alternatives.

Recently, numerous approaches have been proposed to support the modelling and analysis of security properties in software architecture designs. Here a survey in which the approaches are classified into four broad categories: semi-formal (i.e., mainly using semi-formal methods in the approach), formal (i.e., mainly using formal methods in the approach), integrated semi-formal and formal (i.e., using a combination of semi-formal and formal methods), and aspect-oriented (i.e., security non-functional properties are modelled as aspects) approaches are presented.

7.1 Semi-formal Security Modelling and Analysis Approaches

Unified Modelling Language(UML)[9], a well-known notation, is a language for specifying, visualizing, constructing, and documenting designs of software systems. UML provides graphical notations to express the design of software systems with semi-formal syntax and semantics, and an associated language, the Object Constraint Language (OCL), for expressing logic constraints. UML contains two basic diagram types: structure diagrams and behavior diagrams. Structure diagrams depict the static structure of the elements in the system, including class, composite structure, component, deployment, object, and package diagrams. Behavior diagrams depict the dynamic behavior of the elements in the system, including activity, statechart, use case, communication, interaction overview, sequence, and timing diagrams.

UML semantics are defined using a meta-model that is described in a semi-formal manner using three views: the abstract syntax, well-formedness rules, and modelling element semantics. The abstract syntax is provided as a model described in a subset of UML, consisting of a UML class diagram and a supporting natural language description. The well- formedness rules are provided using the OCL and natural language (i.e., English). The UML meta-model is defined as one of the layers of a four-layer meta-modelling architecture, which includes meta-metamodel, meta-model, model, and user objects. This section presents several approaches, which use UML to model and analyze security non-functional properties.

7.2 MAC-UML Framework

The MAC-UML Framework [10] addresses the issue of incorporating Mandatory Access Control (MAC) into UML design artifacts, including use case, class, and sequence diagrams. The approach focuses on providing support for the definition of clearances and classifications for relevant UML elements.

In this section the concept of security assurance rules for a UML design is presented. The basis of such security assurance rules is that UML use case diagrams, class diagrams, and sequence diagrams are abstracted into a set of UML elements. For example, there is a UML use case set $UC = \{uc_1, uc_2 ...\}$, UML actor set $AC = |ac_1, ac_2 ...\}$, UML class set $C = \{c_1,c_2 ...\}$, and UML method set $M = \{m_1,m_2 ...\}$. Each UML element is assigned a clearance (CLR) or classification (CLS) from the partially ordered set ^ = {± = o_1,a_2 ...,<r_s} where the order relation $a^* < oj(i < j)$ means the security level oj has a higher security concern than that of a^*. Notations *ac.CLR*, *uc.CLS*, *c.CLS*min, *c.CLS*max and *m.CLS* represent the CLR of actor *ac*, the CLS of use case *uc*, the min and max CLS of class *c*, and the CLS of method *m*, respectively. Then, three tiers of MAC security assurance rules are defined to assess the question of how to attain security in a design. Tier 1 security assurance rules represent the creation of use case diagrams with actors, use cases, actor-use case associations, actor and use case

inheritance, and use case inclusion and extension relationships. For example, one of these security assurance rules can be interpreted as: For every actor *ac*m that is associated with the use case *uc** (as a behavior of the application), the CLR of the actor *ac*m must dominate the CLS of the use case uc*. Formally, it can be represented as: V actor *ac*m and use case *u c a c* m is securely (MAC) associated with *uc** ^ *ac*m*CLR* > *ucy.CLS*.

Correspondingly, security assurance rules for actor inheritance, use case inheritance, use case inclusion, and use case extension have also been defined. Tier 2 security assurance rules emphasis on defining the classes that are utilized by each use case: for a class *c* (intended) to be used in a sequence diagram to serve the goal (i.e., realize the functionality) of use case *uc*, the CLS of the *uc* must dominate the minimum CLS of *c*. Tier 3 security assurance rules are a refinement of Tier 2 security assurance rules to support method calls between the different entities (use case and objects) in a sequence diagram. Finally, algorithms are defined for assessing whether a UML design as a whole satisfies the security assurance rules by conducting a comprehensive analysis of the entire design.

The security problem addressed in the approach is mandatory access control (refer to Figure 7.1). The example system used is a Survey Institution which performs and manages public surveys. In the system, after the raw data of the survey is collected, staff with different privileges will manipulate the database, where senior person can add a survey header into the database, and another staff person (senior or junior staff) will add questions into that survey, and also have the ability to categorize questions and add a new question category if needed. However, there are some special questions that have more sensitive content, which only senior staff are allowed to perform data entry. The strength of the approach is that it bridges the gap between software engineering and an organization's security personnel. With the enforcement and assessment of three tiers of security assurance rules, the MAC capability can be incorporated into a UML design, where access violations through inheritance, inclusion, and extension can be

detected. The approach can be applied to systems where MAC is one of the priority concerns. Such systems can be a distributed system, or an information system. The limitation of the approach is: since security assurance rules in the approach are only explored on a subset of UML diagrams (i.e., use case, class, and sequence diagram), the approach is only applicable to a system design that uses these three kinds of diagrams. In addition, security analysis of the approach is based on the semi-formal UML, where the relationship inheritance, inclusion, and extension are not formally defined. To obtain more rigorous analysis results, a formal specification of the system design is desired.

7.3 SecureUML

SecureUML [11] is a modelling language that defines a vocabulary for annotating UML based models with information relevant to access control. It is based on the role-based access control model (RBAC), with additional support for specifying authorization constraints. SecureUML defines a vocabulary for expressing different aspects of access control, like roles, role permissions, and user-role assignments. Due to its general access-control model and extensibility, SecureUML is well suited for business analysis as well as design models for different technologies. The structure of the modelling language conforms to the reference model for model driven systems. Model-driven software development is an approach where software systems are defined using models and constructed, at least in part, automatically from these models. A system can be modelled at different levels of abstraction or from different perspectives. The syntax of every model is defined by a meta-model.

The SecureUML meta-model is defined as an extension of the UML meta-model. The concepts of RBAC are represented directly as meta-model types, including User, Role and Permission as well as relations between these types. Instead of defining a dedicated meta-model type to represent protected sources, every UML model element is allowed to take the role of a protected resource. Additionally, the

type ResourceSet is introduced, which represents a user defined set of model elements. A Permission is a relation object connecting a role to a ModelEle- ment or a ResourceSet. The semantics of a Permission are defined by the ActionType elements used to classify the Permission. Every ActionType represents a class of security relevant operations on a particular type of protected resource, for example, a method with the security relevant action executes, or an attribute with the action changes. ActionTypes give the developer a vocabulary to express permissions at a level close to the domain vocabulary. The set of ActionTypes available in the language can be freely defined using ResourceType elements. A ResourceType defines all action types available for a particular metamodel type. The connection to the meta-model type is represented by the attribute BaseClass, which holds the name of a type or a stereotype. The set of resource types and their action types, and the definition of their semantics on a particular platform, define the resource type model for the platform. An AuthorizationConstraint is a part of the access control policy of an application. It expresses a precondition imposed on every call to an operation of a particular resource, which usually depends on the dynamic state of the resource, the current call, or the environment. AuthorizationConstraint is derived from the UML core type Constraint. Such a constraint is attached either directly or indirectly, via a Permission, to a particular model element representing a protected resource.

The security problem addressed in the approach is role- based access control (refer to Figure 7.1). Enterprise JavaBeans (EJB) has been used in the approach. EJB is used in the industry for developing distributed systems. It is an industry standard with strong security support, which is implemented by a large number of application servers. The access control model of EJB is RBAC, where the protected resources are the methods accessible by the interfaces of an EJB. An access control policy is mainly realized by using declarative access control. This means that the access control policy is configured in the deployment descriptors of an EJB component. The security subsystem

of the EJB environment is responsible for enforcing this policy on behalf of the components. The strength of the approach is that it developed a modelling language to build on the access control model of RBAC, with additional support for specifying authorization constraints in the context of a model-driven software development process to generate access control infrastructures. The approach helps to realize the RBAC capabilities in UML. The approach is suitable for distributed systems that incorporate RBAC model, such as an online banking system. However, the limitation of the approach is: currently, the approach only focuses on UML static design models, which are relatively close to the implementation.

It is worth considering whether the efficiency of the development process of secure applications can be improved by annotating models at a higher level of abstraction (e.g. analysis) or by annotating dynamic models, e.g. UML state machines.

7.4 Separating Modelling of Application and Security Concerns

Separating Modelling of Application and Security Concerns (SMASC) is an approach proposed to model system's functional requirements separately from security requirements using UML use case, class, and object collaboration diagrams [12]. The motivation of the approach is to make a secure application system more maintainable with minimal impact on application concerns from changes to security concerns or vice versa. In the approach, the system is viewed through multiple views: a functional requirement view in UML use case diagram; a static view in UML class, and a dynamic view in UML object collaboration modelling. The system's functional requirements are modelled in "non-secure" business use cases. Security concerns are captured in security use cases and security objects, where security use cases are realized through message communications among security objects. The security use cases and objects can also be used in different application systems. Similarly, in the static model of the system, security concerns are separated from business concerns by modelling

non-secure application classes separately from security service classes, and in the dynamic model, security concerns are separated from business concerns by modelling non-secure application objects separately from security objects. Therefore, the system's use cases, classes, and objects are divided into business layer and security layer. Security requirements are considered optional features meaning that if the security feature is required in a given application, then the appropriate security requirement condition is set to true, otherwise it is set to false. When the system requires security services, the security use cases are extended from the non-secure business use cases at extension points, which is a location where a use case extends another use case if the specified condition holds. The security use cases can have parameters, whose values are passed from the business use cases that they extend. Consequently, security classes and objects that realize the security feature can be added into the system's static and dynamic model.

The security problem addressed in the approach include integrity and non-repudiation (refer to Figure 7.1). The example system is an e-commerce application, where security concerns are separated from business concerns by modelling non-secure e-commerce application classes in the e-commerce business layer and secure classes in security layer. The strength of the approach is that it provides a way to capture security requirements in security use cases and encapsulate such requirements in security objects separately from the application requirements and objects. The approach reduces system complexity caused by mixing security requirements with business application requirements, thus to increase the system's maintainability and components reusability. However, one issue of the approach is that usually, security property is a pervasive property for a system which may crosscut many design model elements; therefore, clear separation of business and security model elements would not be a trivial task in this object-oriented approach.

7.5 Formal Security Modelling and Analysis Approaches

Formal methods [13] are referred to the variety of mathematical modelling techniques that are applicable to specify, develop, and verify computer system (software and hardware) design. A system's formal specification is the use of notations derived from formal logic to describe assumptions about the world in which a system operates, requirements that the system is to achieve, and a design to meet those requirements. Formal methods provide a way that a system can be formally specified whereby its desired properties can be reasoned about rigorously. Formal methods have been used to represent software architectures, where they provide a formal, abstract model for systems; thus, a means of describing and analyzing software architectures and architectural styles is available. This section presents several approaches that use formal methods to model and analyze security properties.

Software Architecture Model

Software Architecture Model (SAM) [14] is a general formal framework for visualizing, specifying, and analyzing software architectures. In SAM, a software architecture is visualized by a hierarchical set of boxes with ports connected by directed arcs. These boxes are called compositions. Each composition may contain other compositions. The bottom-level compositions are either components or connectors. Various constraints can be specified. This hierarchical model supports compositionality in both software architecture design and analysis, and thus facilitates scalability. Each component or connector is defined using Petri net. Thus, the internal logical structure of a component or connector is also visualized through the Petri net structure. Textually, a SAM software architecture is defined by a set of compositions $C = C\backslash, C_2,..., C_k$ (each composition corresponds to a design level or the concept of sub-architecture). Each composition $Ci = \{Cmi, C_{ni}, Csi\}$ consists of a set C_{mi} of components, a set C_{ni} of connectors, and a set C_{si} of composition constraints. An element Cij =

*(S*ij, *B*ij), (either a component or a connector) in a composition C_i has a property specification Sij (a temporal logic formula) and a behavior model B_ij (Petri net). Each composition constraint in C_{si} is also defined by a temporal logic formula. The interface of a behavior model Bij consists of a set of places (called ports) that is the intersection among relevant components and connectors. Each property specification Sij only uses the ports as its atomic propositions/predicates that are true in a given marking if they contain appropriate tokens. A composition constraint is defined as a property specification, however it often contains ports belonging to multiple components and/or connectors. A component Cij can be refined into a lower-level composition *C*i, which is defined by *h(C*ij) = C;(h is a hierarchical mapping relating compositions). The behavior of an overall software architecture is derived by composing the bottom-level behavior models of components and connectors. SAM provides both the modelling power and flexibility through the choice of different Petri net models. In SAM, software architecture properties are specified using a temporal logic. Depending on the given Petri net models, different temporal logics are used.

In [15], SAM is applied to support the formal design of software architecture for secure distributed systems. The security problem addressed is general information confidentiality (refer to Figure 7.1). The Petri net model used is the Predicate Transition Nets. The linear-time temporal logic is selected to formally specify security policies, the Chinese Wall policy, where *Basic objects* are individual items of information (e.g. files), each concerning a single corporation; *Company datasets* define groups of objects that refer to the same corporation; *Conflict of interest classes (COI)* define company datasets that refer to competing corporations. Subsequently, the definition of the sensitive dataset and secure distributed control architecture are provided. Finally, a new concept called the dependence relation for the Petri net model is defined, which gives source and sink of every work flow in the model and the dominating elements. Given an architecture model of a distributed system, a set of rules have been given to reconstruct the software architecture and enforce the security policy in the workflow

level for a software architecture by examining the flow of sensitive datasets between tasks.

The strength of the approach is that it integrates two formalisms, Petri nets and temporal logic, to specify software architectures. The properties of the software architecture (e.g., information flow, deadlock, etc.) specified in Petri nets can be proved using temporal logic. Consequently, model checking techniques can be employed to realize the automated verification of software architecture properties. In addition, the approach provides a hierarchical architecture specification model, which enables iterative model checking in a bottom-up fashion. However, one issue of the approach is because of the limitation of model checking, the approach is generally not applicable to infinite state systems.

Multi Security Level Architecture

A modelling method for the Multilevel Security (MLS) architecture of the Defense Advanced Research Projects Agency's (DARPA's) Polymorphous Computing Architecture (PCA) program is proposed in [16]. PCA is a multi-processor architecture that allows a processor to morph during operations to provide the best type of processor for the job at hand. The goal of MLS-PCA is to create a high assurance security infrastructure for multiprocessor distributed applications, which means that the trusted aspects of the system needs to be verified, at a high level, under a certification program, such as the DoD's Trusted Computer Security Evaluation Criteria or its replacement, the Common Criteria (CC). In the proposed architecture, each single level Avionics Application Process (AAP) is front-ended by an Encryption Processing Element (EPE). All communication by an AAP must go through an EPE. All communication between EPEs is encrypted and authenticated. Keys are distributed to the EPEs by the Network Security Element (NSE) based on a security policy set up by mission control. The NSE enforces both Mandatory Access Control (MAC) and Discretionary Access Control (DAC). There is a unique key for each element of the security policy. For example, there is a key for each security level and

compartment in the MAC security lattice, as well as for each pair in the DAC matrix. The NSE generates a session key between two AAPs by XORing the relevant policy keys with a onetime random key. The session key is then distributed to each of the AAP's EPE, where this session key must be distributed encrypted. All connections between two AAPs are simple. This allows a low level process to send information up to a high level process, but not vice versa. Another type of connection, called a coalition, that consists of AAPs at a common security level and using a common key is also allowed. In addition, the EPEs are also transparent to the AAPs, preventing the EPEs themselves from being used as a covert channel. High levels of evaluation require formal models and analysis. The selected formal method is Alloy [17]. The language is based on set theory and the first-order logic, similar to Z, with the standard set operators and quantifiers. A state is defined by sets and relationships among them. An operation will transform a state to a new state, i.e., the sets are modified. Alloy also allows the specification of invariants.

The security problem addressed in the approach is authentication (refer to Figure 7.1). With the use of formal method Alloy and its analysis tool in the approach, one can check the correctness of software architecture specification, using an inductive argument to claim that if an initial state is legal and all operations produce legal states, the system cannot be in an illegal state and the specification is correct. The approach also can be used to determine if a software architecture specification is overconstrained or under-constrained. The approach forces designers to look at the details of the architecture at an early stage of development, thus problems are detected and verification provides evidence that the requirements are maintained. One issue of the approach is: as in the analysis of an Alloy specification, a solution is obtained in a specific scope, therefore, the analysis of the approach is correct, but not complete.

Security Check

Security check is a technique proposed in [18] to entail taking small units of a system, putting them in a "security harness" that exercises relevant executions appropriately within the unit, and then model checking these tractable units. The technique is inspired by unit verification. The basic semantic framework used in the modelling is *discrete time labeled transition systems*. A discrete-time transition labeled transition system (DTLTS) is a tuple < *S,A* sj > where: *S* is a set of states; *A* is a visible action set; ^ is the transition relation; and sj is the start state. The distinguished elements t and *i* correspond to the *internal action and clock-tick* (or *idling)* action. A DTLTS encodes the operational behavior of a real time system. States may be seen as "configurations" the system may enter, while actions represent interactions with the system's environment that can cause state changes. The transition relation records which state changes may occur: if < *s, a, s'* > is a transition relation, then the transition from state *s* to *s'* may take place whenever action *a* is enabled. t is always enabled; other actions may require "permission" from the environment in order to be enabled. Also, transitions except those labeled by *i* are assumed to be instantaneous. While unrealistic at a certain level, this assumption is mathematically convenient, and realistic systems, in which all transitions "take time", can be easily modelled. If a DTLTS satisfying the maximal progress property is in a state in which internal computation is possible, then no idling (clock ticks) can occur. DTLTSs model the passage of time and interactions with a system's environment. Finally, DTLTSs may be *minimized* by merging semantically equivalent but distinct states. The properties prove in *security check* are safety properties, including quasiliveness or bounded response which is a reasonable weakening of classical liveness. Both these classes of properties are inherent in any security property specification. While safety deals with properties of the form "nothing bad will happen" (i.e., the private key can never be revealed), liveness deals with assured- response or in a temporal setting bounded-response

(i.e., the system will always respond in “t” time units even when under a DOS attack). *Security check* works by taking the property to be proved on the system and suitably crafting a “test process” based on that property (safety or liveness). The “unit”, or modules inside the system to which the property is applicable, is isolated, all the behavior of the process not relevant to the property in question is “sealed” off and this transformed “unit” is first minimized and then run in parallel with the test process. Then it is checked if the test process terminates by emitting a pre-designated “good” or a “bad” transition. Depending on the transition the test process emitted it can be decided if a property is satisfied by the system or not.

The security problem addressed in the approach is to improve the intrusion detection capabilities for distributed system. The security check approach offers a means of coping with state explosion of a system. The approach also enables to detect system vulnerabilities even when the attack behavior is not known. And for known attack patterns the approach can provide models of suspicious behavior which can then be used for intrusion detection at a later stage. One issue of the approach is since modelling checking techniques have been used; the approach is more suitable to finite state systems.

CVS-Server Security Architecture

An outline of a formal security analysis of a CVS-Server architecture is presented in [19]. The analysis is based on an abstract architecture (enforcing a role-based access control on the repository), which is refined to an implementation architecture based on the usual discretionary access control provided by the POSIX environment). The *Concurrent Versions System* (CVS) is a widely used tool for version management in many industrial software development projects, and plays a key role in open source projects usually performed by highly distributed teams. CVS provides a central database (the *repository*) and means to synchronize local modifications of partial copies (the *working copies*) with the global repository. CVS can be accessed via a network; this requires a security architecture establishing authentication, access

control and non-repudiation. The proposed architecture aims to provide an improved CVS server, which overcomes the shortcomings of the default CVS server. The first aim of the work is to provide a particular configuration of a CVS server that enforces a role-based access control security policy. The second aim is to develop an "open CVS-Server architecture", where the repository is part of the usual shared file system of a local network and the server is a regular process on a machine in this network. While such an architecture has a number of advantages, the correctness and trustworthiness of the security mechanisms become a major concern, which leads to use formal methods to analyzing the access control problem of complex system technology and its configuration. The formal method Z has been chosen as the specification formalism. The modelling and theorem proving environment Isabelle/HOL-Z 2.0 is used, which is an integrated documentation, type- checking, and theorem proving environment for Z specifications.

The security problem addressed in the approach is role- based access control (refer to Figure 7.1), which is modelled and analyzed in the context of a CVS system. Therefore, the RBAC addressed in the approach is not generic to other applications.

7.6 Integrated Semi-formal and Formal Modelling and Analysis Approaches

This section presents the approaches that use a combination of semi-formal and formal methods to model and analyze security non-functional properties.

UML/Theorem Prover Approach

An extensible verification framework for verifying UML models for security requirements is presented in [20]. In particular, it includes various plug-ins performing different security analysis on models of the security extension UMLsec. Then an automated theorem prover binding is used to verify security properties of UMLsec models that make use of cryptography (such as cryptographic protocols). The UMLsec is an

extension to UML that allows the expression of security relevant information within the diagrams in a system specification. UMLsec is defined in the form of a UML profile using the standard UML extension mechanism. The analysis routine in the verification framework supports the construction of automated requirements analysis tools for UML diagrams. The framework is connected to industrial CASE tools using data integration with XMI and allows convenient access to this data and to the human user. Advanced users of the UMLsec approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes, in a way that allows them to concentrate on the verification logic (rather than on user interface issues). The usage of the framework proceeds as follows: the developer creates a model and stores it in the UML 1.4/XMI 1.2 file format; the file is imported by the UML verification framework into the internal Metadata Repository (MDR). MDR is an XMI-specific data-binding library that directly provides a representation of an XMI file on the abstraction level of a UML model through Java interfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, statecharts, and stereotypes. Each plug-in accesses the model through the JMI interfaces generated by the MDR library. The plug-ins may receive additional textual input and they may return both a UML model and textual output. The plug-ins include static and dynamic checkers. The static checker parses the model, verifies its static features, and delivers the results to the error analyzer. The dynamic checker translates the relevant fragments of the UML model into the automated theorem prover input language. The automated theorem prover is spawned by the UML framework as an external process; its results are delivered back to the error analyzer. The error analyzer uses the information received from the static checker and dynamic checker to produce a text report for the developer describing the problems found, and a modified UML model, where the errors found are visualized. Besides the automated theorem prover binding presented in this chapter there are

other analysis plugins including a model-checker binding and plug-ins for simulation and test-sequence generation.

The security problem addressed in the approach is authentication (refer to Figure 7.1). The approach has been applied to a security-critical biometric system, where the control access to protected resources, such as a user's biometric reference data, needs to be ensured. Therefore, a cryptographic protocol is needed to protect the communication between the user biometric data reader and the host system. The protocol uses message counter in the transmission messages, thus to detect attacks. With the application of the approach, a flaw in the protocol, which allows attackers to misuse those message counters, has been detected. However, security properties that can be analyzed in the approach are limited to those which can be represented in first-order logic.

UML/Promela Approach

The UML/Promela approach [21] is proposed to investigate an appropriate template for security patterns that is tailored to meet the needs of secure systems development. In order to maximize comprehensibility, the well-known notation UML is used to represent structural and behavioral information. Furthermore, the verification of security properties is enabled by adding formal constraints to the patterns. The enhanced *security pattern template* presented herein contains additional information, including behavior, constraints, and related security principles, that addresses difficulties inherent to the development of security-critical systems. The security needs of a system depend highly on the environment in which the system is deployed. By introducing and connecting general security properties with a pattern's substance, the developer can gain security insight by reading and applying the pattern. Furthermore, behavioral information and security-related constraints are included in the security pattern template. The developer can use this information to check if a specific design and implementation of the pattern is consistent with the essential security properties. The augmented security pattern template includes

fields' applicability, behavior, constraints, consequences, related security pattern, supported principles, and thus enhances the communication of security-specific knowledge that is related to a concrete application. Finally, a UML formalization framework that is developed to support the generation of formally specified models defined in terms of Promela [22], the language for the Spin model checker, thus to analyze the security pattern related requirements.

The security problem addressed in the approach is authorization (refer to Figure 7.1). The approach has been applied to an example system, where security properties, such as access violations from external requests to the system's internal entities, can be verified. These properties are instantiated in terms of linear time temporal logic to enable the analysis with Spin model checker. However, the approach currently focuses on the security property analysis against requirements. It needs to be extended in order to support the architecture level design and analysis of security properties.

7.7 Aspect-Oriented Security Modelling and Analysis Approaches

The principle of separation of concerns has long been a core principle in software engineering. It helps software engineers with managing the complexity of software system development. This principle refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concern (concept, goal, purpose, non-functional properties, etc.). However, many concerns of a system tend to crosscut many design elements at the design level; their implementation tends to crosscut many code units. Aspect-Oriented Software Development (AOSD) [23] technologies have been proposed to enable the modularization of such crosscutting concerns. In AOSD, a system's tangling concerns or pervasive properties are encapsulated in model element called an aspect. Subsequently, a weaving process is employed to compose core functionality model elements with these aspects, thereby generating an architecture design. This section presents aspect-oriented approaches

which have been proposed to model and analyze security nonfunctional properties.

Aspect-Oriented Secure Application

An experience report based on developing security solutions for application software is presented in [24]. The programming language AspectJ [25] is used for this purpose. The engineering of application level security requirements are considered in this report, where the security concern covers many aspects, including authentication, auditing, authorization, confidentiality, integrity and non-repudiation. Security is a pervasive requirement for an application. Modularizing security concerns is a difficult task, and where and when to call a given security mechanism in an application has not been addressed adequately either. Furthermore, the crosscutting nature of security not only relates to the diversity of specific places where security mechanisms are to be called: some security mechanisms also require information that is not localized in the application. An example used in the report describes a Personal Information Management (PIM) system. A PIM system supports the human memory by keeping track of personal information, including a person's agenda, contact information of friends and business contacts, the tasks he has to fulfill, etc. A palm pilot is a typical example of a PIM system. In this system, the security requirement is the enforcement of access control. The design of this application is captured in a UML class diagram, where a class called PIMSystem is the center of the model. Through this class, the system can represent and manage three different types of information: appointments, contacts and tasks. Two security access rules are: for appointments and tasks, the owner can invoke all their operations; other persons are only allowed to view them; for contacts, only owners can perform their operations. Finally, the report focuses on implementing these rules as aspects in AspectJ.

The security problem addressed in the approach is authorization (refer to Figure 7.1). The approach provides a way to implement authorization properties as crosscutting concerns in the aspect-oriented

programming language AspectJ. However, it did not address the problem of aspect-oriented design of security properties.

Formal Design Analysis Framework

Formal Design Analysis Framework (FDAF) [26, 27, 28], is an aspect-oriented approach proposed to support the design and automated analysis of non-functional properties for software architectures. In the FDAF, non-functional properties are represented as aspects. At the architecture design level, aspect represents either a property that permeates all or part of a system, or a desired functionality that may affect the behavior of more than one architecture design elements, such as security aspects. Security aspects, including data origin authentication, role-based access control, and log for audit, have been defined in the FDAF. The definition of these security aspects uses UML diagrams. The definition for a security aspect includes its static view and dynamic view. The static view of a security aspect is defined using UML class diagram, presenting the attributes and operations need to be used in order to include the desired functionality in a system. It may also include OCL invariants, pre-conditions, and post-conditions regarding the weaving constraints of the security aspect. The dynamic view of a security aspect is defined in UML sequence diagram, describing the dynamic behavior of the security aspect, including the information about when and where to use the operations defined in the security aspect's static view. The FDAF proposes a UML extension to support the modelling of security aspects in UML. This extension assists architects in weaving an aspect into a design and updating an aspect in a design. The syntax and semantics for this UML extension have been defined. The FDAF uses a set of existing formal analysis tools to automatically analyze a UML architecture design that has been extended with security aspects. Architecture designs are documented using extended UML class diagram and swim lane activity diagram in the FDAF. The automated analysis of an extended UML architecture design in existing formal analysis tools is achieved by formalizing part of UML into a set of formal languages that have tool support. The

translation into a formal language with existing tool support leverages a large body of work in the research community. The formalization approach used is the translational semantic approach [29]. In translational semantics, models specified in one language are given semantics by defining a mapping to a simpler language, or a language, which is better understood. Algorithms for mapping UML class and swim lane activity diagrams to a set of formal languages have been defined, verified with proofs, and implemented in the FDAF tool support, thus automated translation from UML to this set of formal languages are realized. Formal languages that UML can be formalized in the FDAF for the analysis of security properties include Promela [22] and Alloy [17], where Promela's analysis tool is used to analyze data origin authentication and log for audit security aspect design and Alloy's analysis tool is used to analyze role-based access control security aspect UML design.

The example system selected in the FDAF is the Domain Name System (DNS) [30], which is a real-time, distributed system. The security problem addressed includes data origin authentication, role-based access control, and log for audit (refer to Figure 7.1). There three security aspects have been modelled in the DNS, where data origin authentication is used to ensure the source of data, role-based access control is used to ensure the access control of the DNS database, and log for audit is used to log DNS messages. The strength of the approach is it integrates the well-known semi-formal notation UML and a set of existing formal notations into an aspect-oriented architectural framework to support the design and analysis of non-functional properties, such as security and performance. A limitation of the approach is that the analysis of a UML based architecture design uses existing formal tools, the limitations of these tools affect the analysis results provided in terms of accuracy, useful analysis data extraction, and interpretation of the results.

7.8 Discussion

A summary of the approaches in the survey is presented in Table 7.1. The approaches in this survey support modelling of one or more security properties at the architecture design level; many also support their automated analysis. The comparison criteria defined for this survey are: identifying the specific security property addressed (e.g., role-based access control, authentication, etc.), modelling notation(s) used, whether or not automated security property analysis is supported, and the kind of example system the approach has been applied to. Each of these criteria is useful, as the results can be used to guide the selection of an appropriate approach and identify possible areas for future research. For example, if one needs to model (but perhaps not analyze) role-based access control for a distributed system, then UML can be selected as the modelling notation. However, if there is a need for a more rigorous, automated analysis of the security property, then a formal method would be more suitable, such as Z or Alloy. If one needs to model confidentiality for information system, then Petri nets and temporal logic are candidate notations, as these have been already been successfully used to model this property. This is not to say that other notations could not be used to model confidentiality. Actually, it opens a wide variety of possible future research topics that investigate the use of different notations, tailored notations, and perhaps identifying a set of notations that are suitable for modelling a comprehensive collection of security properties. It is also important to note that the validation of the approaches presented in the literature has typically been made using one example system. Additional validation of the approaches used to model and analyze security properties for the architecture design is necessary.

Table 7.1: Overview of approaches to design and analyze security properties.

	Security Property	Modelling Notations	Analysis	Example System
UML-MAC Framework	Mandatory Access Control	UML	Supported	Information System
SecureUML	Role-Based Access Control	UML	Not supported	Distributed System Using EJB
SMASC	Integrity, NonRepudiation	UML	Not supported	E-Commence Application System
Software Architecture Model	Confidentiality	Petri Nets, Temporal Logic	Not supported	Information System Using Chinese Wall Policy
Multi Level Security Architecture	Authentication	Alloy	Automated analysis	Real-Time System: MLS-PCA
Security Check	Intrusion detection	Discrete Time Labeled Transition	Automated analysis	Distributed System
CVS-Server Security Architecture	Role-Based Access Control	Z	Automated analysis	Distributed system: CVS
UML/ Theorem Prover Approach	Authentication	UML, First-Order Logic	Automated analysis	Biometric system
UML/Promela Approach	Authorization	UML, Liner Time Temporal	Automated analysis	Distributed System

Aspect Oriented Secure Application	Authorization	UML	Not supported	Information System: Personal Information Management
FDAF	Data Origin Authentication, Role-Based Access Control, Audit	UML, Formal languages (Promela, Alloy)	Automated analysis	Real-Time, Distributed System: Domain Name System

Advancement questions

1. What are the main components of the security?
2. What is the main purpose of the The MAC-UML Framework?
3. What the modelling language SecureUML is developed for?
4. What is the on the role-based access control model?
5. What is the main idea of the Separating Modelling of Application and Security Concerns approach?
6. What are the main features of the Software Architecture Model?
7. How can we model and analyze the security non-functional properties?
8. What are tha main profirs in the usage of the UML/Promela approach?
9. What are the main intended purpose of the AspectJ programming language?
10. What the Formal Design Analysis Framework is design for?

REFERENCES

[1] Lirong Dai A Survey of Modelling and Analysis Approaches for Architecting Secure Software Systems / Lirong Dai, Kendra Cooper // International Journal of Network Security. – 2007. - Vol.5, No.2. - PP.187–1987.

[2] Arbaugh B. Security: Technical, social, and legal challenges /B. Arbaugh // Computer. - Feb. 2002. -vol. 35. - issue 2. - pp. 109-111.

[3] Davis N. Processes for producing secure software/ N. Davis, W. Humphrey, S. T. Jr. Redwine, G. Zibul- ski, and G. McGraw // IEEE Security & Privacy Magazine.- May-June 2004 . -vol. 2. -no. 3.- pp. 18-25.

[4] Saltzer J. The protection of infor¬mation in computer systems/ J. Saltzer and M. Schroeder // Proceedings of IEEE.- 1975 .- vol. 63.- no. 9.- pp. 1278-1308.

[5] Computer Emergency Readiness Team Coordination Center (CERT/CC) [Electronic resource] : 2004 E-Crime Watch. - Access mode:, http: //www.cert.org/about/ecrime.html – Name from screen

[6] The Information Security Standard: Information technology : ISO 17799. - Code of practice for information security management, 2000.- ISO Standart

[7] Balsamo S. Experimenting different software architectures performance techniques: A case study/ S. Balsamo, M. Marzolla, and A. D. Marco// in Proceedings of the Fourth International Workshop on Software and Performance.- 2004 .- pp. 115-119.

[8] Software Architecture: Per¬spectives in an Emerging Discipline/ M. Shaw and D. Garlan.- Prentice Hall.- 1996.

[9] The Unified Modelling Language Reference Manual 2nd Edition/ J. Rumbaugh, I. Jacobson, and G. Booch.- Addison-Wesley, Reading MA.- 2004.

[10] Doan T. MAC and UML for secure software design / T. Doan, S. Demurjian, T. C. Ting, and A. Ket- terl// in Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering.- 2004. - pp. 75-85.

[11] Lodderstedt T. Se- cureUML: A UML-based modelling language for model-driven security / T. Lodderstedt, D. Basin, and J. Doser // in Proceedings of the 5th International Conference on The Unified Modelling Language. - 2002 . - pp. 426-441.

[12] Gomaa H. Modelling complex systems by separating application and security concerns. / H. Gomaa and M. Eonsuk Shin // in Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems. -2004. - pp. 19-28.

[13] Bowen J. P. An invitation to formal methods / J. P. Bowen, R. W. Butler, D. L. Dill, R. L. Glass, D. Gries, and A. Hall.// Computers. - 1996. - vol. 29, issue 4. - pp. 16-29,.

[14] He X. Formally analyzing software architectural specifications using SAM / X. He, H. Yu, T. Shi, J. Ding, and Y. Deng // Journal of Systems and Software/- 2004. - vol. 71, no. 1-2. - pp. 11-29.

[15] Yu H. Formal software architecture design of secure distributed systems / H. Yu, X. He, S. Gao, and Y. Deng // in Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03). - 2003. - vol. 1. - pp. 450-457.

[16] Hashii B. Lessons learned using alloy to formally specify MLS-PCA trusted security architecture / B. Hashii // in Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering.- 2004. - vol. 1. - pp. 86-95.

[17] Software Design Group, the Alloy Analyzer [Electronic resource]: 2002 – 2005. - Access mode: http://alloy.mit.edu.

[18] Ray A. Security check: a formal yet practical framework for secure software architecture / A. Ray // in Proceedings of the 2003 Workshop on New Security Paradigms.- 2003. - vol. 1. - pp. 59-65.

[19] Brucker A. A case study of a formalized security architecture / A. Brucker, and B. Wolff // Electronics Notes in Theoretical Computer Science. - 2003. - vol. 80. - pp. 1-18.

[20] Jurjens J. Sound methods and effective tools for model-based security engineering with UML /J. Jurjens // in Proceedings of the 27th International Conference on Software Engineering.- 2005. - vol. 1. - 322-331.

[21] Using Security Patterns to Model and Analyze Security Requirements, Technical Report MSU-CSE- 03-18 / B. Cheng, S. Honrad, L. Campbell, and R. Wasser- mann. - Department of Computer Science, Michigan State University, 2003.

[22] The Spin Model Checker: Primer and Reference Manual / G. J. Holzmann. - Addison-Wesley, 2003.

[23] Aspect- Oriented Software Development / R. Filman, T. Elrad, S. Clarke, and M. Aksit. - Addison Wesley Professional, 2005.

[24] 30 Win B. Developing secure applications through aspect-oriented programming / B. Win, W. Joosen, and f. Piessens // Aspect-Oriented Software Development (ISBN: 0321219767). - 2005. - pp. 633-651.

[25] AspectJ in Action: Practical Aspect- Oriented Programming / R. Laddad. - Manning Publications, 2003.

[26] Cooper K. Performance modelling and analysis of software architectures: an aspect-oriented UML based approach / K. Cooper, L. Dai, and Y. Deng // Journal of Science of Computer Programming,

System and Software Architectures. - July 2005. - vol. 57, no. 1. - pp. 89-108.

[27] Formal Design Analysis Framework: An Aspect-Oriented Architectural Framework / L. Dai. - The University of Texas at Dallas, Ph.D. Dissertation, 2005.

[28] Dai L. Modelling reusable security aspects for software architectures: a pattern driven approach / L. Dai, K. Cooper, and E. Wong // in Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering. - July 2005. - pp. 163-168.

[29] Hofstede A. H. M. How to formalize It? Formalization principles for information system development methods / A. H. M. Hofstede, and H. A. Proper // Information and Software Technology. - 1998. - vol. 40, no. 10. - pp. 519-540,.

[30] Domain Names - Concepts and Facilities/ P. V. Mockapetris. - IETF STD0013. - November, 1987

CHAPTER 8. FORMAL METHODS FOR ASSURING SECURITY OF PROTOCOLS

Content of the CHAPTER 8

Introduction

Security is an intricate property that is achieved by a combination of sufficiently strong cryptographic algorithms and protocols, correct implementation of hardware and software, and appropriate assumptions about trusted author-ities. Assuring that all of these factors are present and correctly integrated to form a secure system is difficult — if not impossible—without the use of rigorous and formal analytical techniques.

The purpose of this section is to give an overview of some formal methods whose goal is to lend assurance that a system using cryptographic protocols will behave securely. These methods rely on mathematical logic and are accessible to engineers. Because security must be addressed from a variety of viewpoints, it is important to use a variety of methods, each suited for addressing a particular viewpoint.

This section is focused on a single key-distribution protocol—the Needham–Schroeder public-key protocol [1,2] —and highlight how three different formal methods can be used to analyse it. Each method is suited for reasoning about a different aspect of this protocol. Two of the methods— the BAN belief logic [3] and a logic for authentication [4]—are special-purpose logics designed specifically for reasoning about certain security-related properties. The third method—the process algebra CSP [5]—is a general-purpose language for describing and reasoning about protocols with emergent behaviour. Taken together, these systems highlight the very subtle nature of security properties and the need for a variety of views into even a single protocol. They also illustrate how formal methods can identify weaknesses and hidden assumptions underlying security protocols.

8.1 Security primer

Principals are people, keys, processes or machines that send and receive information and that access information resources (databases, processors, printers, etc.). Security properties describe the ability of

principals to access information or resources. Key security properties include:

- *privacy or confidentiality:* knowing with accuracy which principals can read data;
- *integrity:* detecting corruption of information;
- *authentication:* knowing the identity of a principal or the source of information;
- *access control:* restricting the use of a resource to privileged principals;
- *non-repudiation:* preventing principals from subse-quently denying their actions;
- *availability of service:* guaranteeing authorized princi-pals always have access to services.

The *Handbook of Applied Cryptography* [6] describes each of these properties in more detail.

Public-key infrastructure

In public-key cryptography, encryption and decryption are provided through pairs of related keys: each principal has both a *public key* (which is made known to others) and a *private key* (which should be known only to the principal). These keys act as mutual inverses: anything encoded with one of these keys may be decoded with the other. Thus public-key cryptography supports both privacy and digital signatures.

If principal *A* wishes to send a secret message *M* to principal *B*, she encrypts *M* with *B*'s public key: intuitively, only *B* has knowledge of his private key and hence only *B* can decrypt the message. In contrast, if *A* wishes to *sign* a message *M*, she encrypts it with her own *private* key: intuitively, anyone with access to her public key can verify her signature, but no one should be able to forge her signature without knowledge of her private key.

A public-key infrastructure (PKI) supports the distribution, management and use of public keys and certificates to provide authentication, privacy and other security properties. PKIs are based on

certification authorities that vouch for the integrity of cryptographic information and they form the basis of current Internet security.

Basic security foundations

A system's overall security depends on several items, including:

- the cryptographic strength of the system (e.g., the computational infeasibility of decrypting messages without the proper keys);
- the protocols built on top of the cryptographic algorithms (e.g., the secure sockets layer (SSL) protocol used by web browsers);
- the correct association of specific cryptographic keys with specific principals.

Cryptographic strength is assessed over time by a combination of complexity analysis and resistance to cryptanalysis [7]. The use of formal methods to analyse cryptographic strength is not addressed here.

Assessing security protocols often involves an analysis of ways to defeat a protocol by using bits of previously successful protocol runs (known as replay attacks) or by some form of impersonation (e.g., 'man in the middle' attacks). These analyses are particularly important, because these protocols are run repeatedly over time and typically over public networks: many security protocols depend critically on the freshness of secrets.

Associating cryptographic keys with principals is typi-cally done using certificates that are digitally signed by recognized certification authorities, as in the X.509 public-key certificate standard [8]. Determining the public key of a principal is done using a chain of certificates that enable one principal to move from one certificate authority to another in a secure (i.e. digitally signed and checked) way to fetch and check the certificate of another principal. The network of certification authorities is referred to as a *trust model* or *trust topology*.

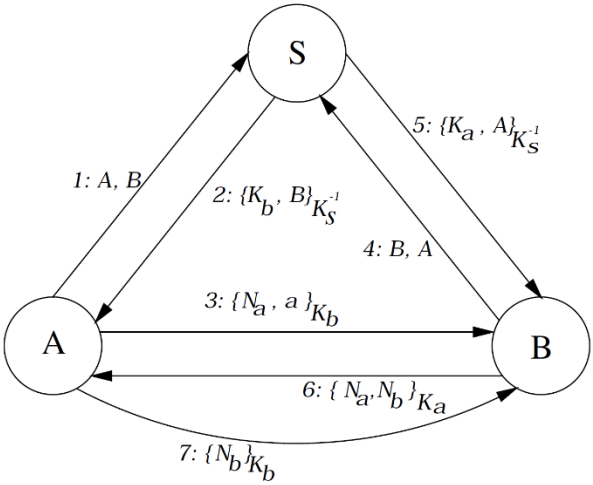


Figure 8.1. Needham-Schroeder protocol.

1. $\mathbf{A} \rightarrow \mathbf{S}$: $\mathbf{A}, \mathbf{B}$
2. $\mathbf{S} \rightarrow \mathbf{A}$: $\{\mathbf{K_b}, \mathbf{B}\}_{\mathbf{K_s^{-1}}}$
3. $\mathbf{A} \rightarrow \mathbf{B}$: $\{\mathbf{N_a}, \mathbf{A}\}_{\mathbf{K_b}}$
4. $\mathbf{B} \rightarrow \mathbf{S}$: $\mathbf{B}, \mathbf{A}$
5. $\mathbf{S} \rightarrow \mathbf{B}$: $\{\mathbf{K_a}, \mathbf{A}\}_{\mathbf{K_s^{-1}}}$
6. $\mathbf{B} \rightarrow \mathbf{A}$: $\{\mathbf{N_a}, \mathbf{N_b}\}_{\mathbf{K_a}}$
7. $\mathbf{A} \rightarrow \mathbf{B}$: $\{\mathbf{N_b}\}_{\mathbf{k_b}}$

Figure 8.2. Needham-Schroeder message exchanges

8.2 Needham–schroeder protocol

An important class of security protocols is the class of key-distribution protocols that enable secure communication sessions where both parties are active at the same time. Examples of such sessions include secure remote logins, secure file transfers and secure electronic-commerce transactions.

The Needham–Schroeder protocol [2] is designed to allow two principals to mutually authenticate themselves through a series of message exchanges, as a prelude for some secure session. A diagram of the message exchanges appears in Figure 8.1, and Figure 8.2 details the message exchanges in linear form.

In these descriptions, the principals are *A*, *B* and *S*, where *S* is a certification authority (*CA*) recognized by principals *A* and *B*. Their public keys are (respectively) K_a , K_b, and K_s; the private key of *S* corresponding to the public key K_s is K_s^{-1}. N_a and N_b are *nonce* identifiers: a nonce is a fresh value created for the current run of a protocol, which has not been seen in previous runs. Let us use common notation for describing security protocols: $A \rightarrow B$ denotes principal *A* sending a message to principal *B*; comma (‘,’) denotes conjunction or concatenation (e.g., ‘*X, Y* ’ denotes a message containing both *X* and *Y*); and $\{X\}_K$ denotes the message *X* encrypted using key *K*.

Principal *A*—wishing to communicate with principal *B*— must get *B*’s public key from *S* and then convey a nonce N_a to *B* (messages 1–3). *B* then needs to get *A*’s public key from *S* and convey a nonce N_b to *A* in such a way that convinces *A* that she is interacting with *B* (messages 4–6). Finally, *A* must convince *B* that he is interacting with *A* and that *A* has received the nonce N_b (message 7).

Messages 1 and 2—as well as messages 4 and 5— correspond to certificate requests from *A* and *B* to an authority *S* for the other’s public key. As messages 1 and 4 are transmitted in the clear, they have no cryptographic significance. In message 3, the nonce N_a is conveyed to *B*. Nevertheless, *B* cannot know for sure who sent N_a : he must assume that the identifier *A* is correct. In message 6, *B* sends to *A* both nonce N_a and N_b to identify himself as *B* to *A*, as well as to convey the nonce N_b to *A*. The basic notion operating here is that *A* and *B* can identify shared secrets (the nonces N_a and N_b) to each other to convince each of the other’s identity.

This protocol has several weaknesses, which are ad-dressed in the subsequent sections. The principals in the protocol implicitly assume that messages 2 and 5 are fresh and are not replays of messages

containing compromised keys; this situation is analysed on *belief logics*. The protocol is also vulnerable to a 'man in the middle' attack.

8.3 Belief logics

One approach to reasoning about key-exchange protocols is to analyse what principals *believe* about important components and properties of protocols. These components include secret and public keys, encrypted messages, messages combined with secrets, and nonces (objects created for a specific run of a protocol). Important properties include freshness (the property of never having been used in a prior run of a protocol), jurisdiction or authority over keys, and binding of secret and public keys to principals.

The Burroughs, Abadi and Needham (BAN) logic [3] is representative of several belief logics [9, 10] that support reasoning about these properties. It focuses on proving goals such as '*A believes* K_b *is B's public key*'. One of the central concerns that BAN logic addresses is the possibility of *replay attacks*, where messages sent during previously successful protocol runs are re-used or replayed to trick principals into using compromised keys or to dupe principals into thinking an intruder is a legitimate participant.

The notion of time in the BAN logic is very simple. Time is divided into two categories: the *present* (i.e. the *current* run of the protocol) and the *past* (i.e. any protocol run preceding the current run).

Syntax and semantics

Let us present only the subset of BAN logic's syntax and semantics necessary to describe part of the Needham– Schroeder protocol. A complete description of BAN logic appears in [3]. The logic includes the following types of statements.

- P |≡ X, read as 'Principal P believes X'. P behaves as if X is true;
- P ◄ X, read as 'P sees X'. A principal has sent P a message containing X;
- P |~ X, read as 'P once said X'. P at some time (either in the present or the past) believed X and sent it as part of a message;

- P |⇒ X, read as 'P has jurisdiction over X'. Principal P has authority over X and is trusted on this matter.
- # (X), read as 'X is fresh'. X has not appeared in a past run of the protocol. $K|\xrightarrow{K}P$, read as 'P has public key K'. The corresponding private key is denoted by K−1 and assumed to be known only by P;

While there are over 19 inference rules that define the semantics of the BAN logic, the following three inference rules capture the essence of the logic for public-key applications.

(1) *Message-Meaning Rule for Public Keys.* If *P* believes that *Q*'s public key is *K*, and if *P* sees a message *X* encrypted with *Q*'s private key K^{-1}, then *P* believes *Q* once said *X*:

$$\frac{\boldsymbol{P}|\equiv|\xrightarrow{K}\mathbf{Q}\ \mathbf{P}<\{\mathbf{X}\}_{\mathbf{K}-\mathbf{1}}}{\boldsymbol{P}\ |\equiv(\mathbf{Q}\ |\sim\mathbf{X})}.$$

This rule uses the belief in the association of a public key with a principal to establish the source of a message.

(2) *Nonce-Verification Rule.* If *P* believes that *X* is fresh (i.e. is new to the current protocol run) and that *Q* once said *X*, then *P* believes that *Q* believes *X* in the current run of the protocol:

$$\frac{\boldsymbol{P}|\equiv\#\ (\mathbf{X})\ \boldsymbol{P}|\equiv(\mathbf{Q}\ |\sim\mathbf{X})}{\boldsymbol{P}|\equiv(\mathbf{Q}\ |\sim\mathbf{X})}.$$

This rule uses the belief about the freshness of a message (usually based on nonces), coupled with knowing the message source, to establish that a principal uttered a message in the current protocol run.

(3) *Jurisdiction Rule.* If *P* believes that *Q* has authority over *X* and that *Q* believes *X*, then *P* will believe in *X*:

$$\frac{\boldsymbol{P}\left|\equiv\left(\mathbf{Q}\Rightarrow\mathbf{X}\right)\boldsymbol{P}\right|(\mathbf{Q}|\equiv\mathbf{X})}{\boldsymbol{P}\ |\equiv\mathbf{X}}.$$

This rule applies to certification authorities. A principal will adopt an authority's belief if that principal recognizes the certification authority. For example, if $\boldsymbol{P}|\equiv\left(\mathbf{Q}\mid\equiv\xrightarrow{\mathbf{K_b}}\mathbf{B}\right)$ and $\boldsymbol{P}|\equiv\left(\mathbf{Q}\Rightarrow\right.$

$\left(\overset{\mathbf{K_b}}{\rightarrow}\ \mathbf{B}\right)$ then $\boldsymbol{P} \mid \equiv \overset{\mathbf{K_b}}{\rightarrow}\ \boldsymbol{B}$.That is, *P* believes K_b is *B*'s public key when *P* recognizes *Q* as a key certification authority.

Analysis of the protocol

When the Needham–Schroeder protocol is analysed using the BAN logic, two possible weaknesses come to light. In steps 2 and 5 of the protocol, certification authority *S* sends to *A* and *B* (respectively) the messages $\{\boldsymbol{K_b}, \boldsymbol{B}\}_{\boldsymbol{K_s^{-1}}}$ and $\{\boldsymbol{K_b}, \boldsymbol{A}\}_{\boldsymbol{K_s^{-1}}}$; that is, *A* and *B* both receive the public key of the principal with whom they wish to communicate.

After *A* receives $\{\boldsymbol{K_b}, \boldsymbol{B}\}_{\boldsymbol{K_s^{-1}}}$ in step 2, the desired result on *A*'s behalf is $\boldsymbol{A} \mid \equiv \left(\overset{\boldsymbol{K_b}}{\rightarrow}\ \boldsymbol{B}\right)$: believes that $\boldsymbol{K_b}$ is *B*'s public key. From the Jurisdiction Rule, this requires two other beliefs:

$\boldsymbol{A}\mid \equiv (\boldsymbol{S}\ \Rightarrow (\overset{\boldsymbol{K_b}}{\rightarrow}\ \boldsymbol{B}))$ (1)

$\boldsymbol{A}\mid \equiv (\boldsymbol{S}\ \mid \equiv\ (\overset{\boldsymbol{K_b}}{\rightarrow}\ \boldsymbol{B}))$ (2)

That is, *A* must believe that *S* has authority over $\boldsymbol{K_b}$ and that *S* believes $\boldsymbol{K_b}$ is *B*'s public key.

The first belief corresponds to *A* believing that *S* is a certification authority having the proper authority to certify *B*'s cryptographic information. This is a reasonable assumption. Establishing the second belief requires the application of both the Message-Meaning and Nonce-Verification Rules.

It is reasonable to assume that *A* knows *S*'s public key: $\left(\text{i.e. } \boldsymbol{A}\mid \equiv\ (\overset{\boldsymbol{K_s}}{\rightarrow}\ \boldsymbol{S})\right)$ From the Message-Meaning Rule and the receipt of certificate $\{\boldsymbol{K_b}, \boldsymbol{B}\}_{\boldsymbol{K_s^{-1}}}$ from *S* we can establish that $\boldsymbol{A} \mid \equiv (\boldsymbol{S} \mid \sim (\overset{\mathbf{K_b}}{\rightarrow}\ \mathbf{B}))$.

That is, *A* believes that *S* once said that $\boldsymbol{K_b}$ was *B*'s public key. We can then use the Nonce-Verification Rule to establish that $\boldsymbol{A}\mid \equiv (\boldsymbol{S}\mid \equiv (\overset{\mathbf{K_b}}{\rightarrow}\ \mathbf{B}))$, that $\boldsymbol{A}\mid \equiv \#(\overset{\mathbf{K_b}}{\rightarrow}\ \mathbf{B})$. This proviso highlights a potential weakness, as there is nothing in message 2 that corresponds to a nonce:

A must assume that anything with the public key from the authority *S* is fresh. Consequently, the protocol is potentially vulnerable to a replay attack based on reusing old keys.

A similar analysis holds true for *A*'s public key in step 5.

8.4 Process algebras

Process algebras such as CSP [5, 11] and CCS [12] provide a way to describe system behaviour in terms of the *events* (i.e. abstract actions deemed 'observable') that can occur. The collection of events that a process can engage in is known as its *alphabet*, typically denoted by _.

The *trace model* of CSP formally describes a process's behaviour in terms of the set of *traces*—that is, sequences of events—that it can perform. These trace sets provide a basis for comparing, equating and refining processes. Two processes are *trace-equivalent* when they have precisely the same sets of traces. A process *Q refines P* in the trace model (written *P Q*) if every trace of *Q* is also a trace of *P* . Intuitively, if *P* corresponds to a specification of permissible behaviour and *Q* refines *P* , then *Q* is guaranteed to exhibit only permissible behaviours.

Using the trace model and the FDR2 model checker [13], Lowe uncovered a previously unknown flaw in the Needham–Schroeder protocol [14, 15]. An informal description of the approach in this section, first introducing the relevant CSP notation is provided.

CSP syntax and semantics

Informally, CSP processes (ranged over by P) can have the following forms.

- STOP is the process that does nothing;
- $e \rightarrow P$ performs event e and then behaves like *P*;
- *P1* π *P2* non-deterministically chooses to behave like *P1* or to behave like *P2*.
- *P1* $[|X|]$ *P2* is a parallel composition of *P1* and *P2*, where the set of events *X* constrains certain actions. Any event in the set X can occur only if *P1* and *P2* both perform it simultaneously; in

contrast, events in $\sum$ - *X* occur only as independent actions of *P1* or *P2*;

- *P \X* behaves like *P*, but the events in *X* are hidden (i.e. considered unobservable);
- *P [e1 ← e1_, . . . , en ← en_]* behaves like *P*, except that each event ei is relabelled to ei_. As a special case, in *P [e ← e1, e ← e2]*, the event e is replaced by a non-deterministic choice between the events e1 and e2.

Formally, we introduce a family of relations $\overset{a}{\Rightarrow}$ that describe processes' trace behaviour, writing $\boldsymbol{P \overset{a}{\Rightarrow} P'}$ to indicate that process *P* can perform the trace *a* and become the process P' This family of relations can be defined as the smallest relations satisfying the axioms and inference rules in Figure 8.3. The first two rules are primarily bookkeeping rules: the first expresses that every process, by doing nothing (_ denotes the empty trace), remains unchanged; the second reflects that computations' intermediate states can be abstracted away (· denotes sequence concatenation). The remaining rules are syntax-directed and formalize the informal explanations of the different process terms given previously. In particular, there is no explicit rule for STOP, because STOP never does anything.

$$\mathbf{P} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P} \quad \frac{\mathbf{P} \overset{\mathbf{a}}{\Rightarrow} \mathbf{P_1} \;\; \mathbf{P_1} \overset{\boldsymbol{\beta}}{\Rightarrow} \mathbf{P_2}}{\mathbf{P} \overset{\boldsymbol{\alpha}\cdot\boldsymbol{\beta}}{\Longrightarrow} \mathbf{P_2}}$$

$$(\mathbf{e} \rightarrow \mathbf{P}) \overset{\mathbf{e}}{\Rightarrow} \mathbf{P}$$

$$\mathbf{P_1} \sqcap \mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_1} \quad \mathbf{P_1} \sqcap \mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_2}$$

$$\frac{\mathbf{P_1} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_1'} \;\; \mathbf{e} \notin \mathbf{X}}{\mathbf{P_1}[|\mathbf{X}|]\mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_1'}[|\mathbf{X}|]\mathbf{P_2}}$$

$$\frac{\mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_2'} \;\; \mathbf{e} \notin \mathbf{X}}{\mathbf{P_1}[|\mathbf{X}|]\mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_1} \;\; [|\mathbf{X}|]\mathbf{P_2'}}$$

$$\frac{\mathbf{P_1} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_1'} \;\; \mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_2'} \;\; \mathbf{e} \in \mathbf{X}}{\mathbf{P_1}[|\mathbf{X}|]\mathbf{P_2} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P_1'}[|\mathbf{X}|]\mathbf{P_2'}}$$

$$\frac{\mathbf{P} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P'} \;\; \mathbf{e} \notin \mathbf{X}}{\mathbf{P}/_{\mathbf{X}} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P'}} \qquad \frac{\mathbf{P} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P'} \;\; \mathbf{e} \in \mathbf{X}}{\mathbf{P}/_{\mathbf{X}} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P'}}$$

$$\frac{\mathbf{P} \overset{\mathbf{e_i}}{\Rightarrow} \mathbf{P'} \quad \mathbf{P}\left[\mathbf{e_1} \leftarrow \mathbf{e_1'}, \ldots, \mathbf{e_n} \leftarrow \mathbf{e_n'}\right] \overset{\mathbf{e_i'}}{\Rightarrow} \mathbf{P'}\left[\mathbf{e_1} \leftarrow \mathbf{e_1'}, \ldots, \mathbf{e_n} \leftarrow \mathbf{e_n'}\right] \quad \mathbf{e_i} \neq \mathbf{e_i'}}{\mathbf{P}\left[\mathbf{e_1} \leftarrow \mathbf{e_1'}, \ldots, \mathbf{e_n} \leftarrow \mathbf{e_n'}\right] \overset{\mathbf{e_i}}{\Rightarrow} \mathbf{P'}\left[\mathbf{e_1} \leftarrow \mathbf{e_1'}, \ldots, \mathbf{e_n} \leftarrow \mathbf{e_n'}\right]}$$

$$\frac{\mathbf{P} \overset{\mathbf{e}}{\Rightarrow} \mathbf{P'} \quad \mathbf{e} \notin \{\mathbf{e_1}, \ldots, \mathbf{e_n}\}}{\mathbf{P}\left[\mathbf{e_1} \leftarrow \mathbf{e_1'}, \ldots, \mathbf{e_n} \leftarrow \mathbf{e_n'}\right] \overset{\mathbf{e}}{\Rightarrow} \mathbf{P'}\left[\mathbf{e_1} \leftarrow \mathbf{e_1'}, \ldots, \mathbf{e_n} \leftarrow \mathbf{e_n'}\right]}$$

Figure 8.3: Transition relations for CSP

By way of illustration, consider the following simple processes:
$Q \equiv a \rightarrow b \rightarrow c \rightarrow \text{STOP}$
$R \equiv c \rightarrow c \rightarrow \text{STOP}$
The following series of examples illustrates these definitions.
(1) Q has precisely four traces: $\langle \quad \rangle, \langle \boldsymbol{a} \rangle, \langle \boldsymbol{a}, \boldsymbol{b} \rangle, \langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \rangle$.

(2) The traces of Q ıı R are simply those of Q and R: $\langle\ \rangle, \langle a \rangle, \langle a, b \rangle, \langle a, b, c \rangle, \langle c \rangle, \langle c, c \rangle$.

(3) The parallel composition Q [|{b, d}|] R has the following traces: $\langle\ \rangle, \langle a \rangle, \langle c \rangle, \langle a, c \rangle, \langle c, a \rangle, \langle c, c \rangle, \langle a, c, c \rangle$, $\langle c, a, c \rangle, \langle c, c, a \rangle$. Note that Q, denied cooperation from R, is unable to perform its b event.

(4) In contrast, Q [|{c, d}|] R has these four traces: $\langle\ \rangle, \langle a \rangle, \langle a, b \rangle, \langle a, b, c \rangle$. R must delay c events until Q is ready to cooperate; furthermore, R can only perform one c event, because Q provides only one opportunity to do so.

(5) The traces of (Q ıı R)\{b, d} are as follows: $\langle\ \rangle, \langle a \rangle, \langle a, c \rangle, \langle c \rangle, \langle c, c \rangle$.

(6) Finally, (Q ıı R)[b ← a, c ← e, c ← d] has precisely these traces: $\langle\ \rangle, \langle a \rangle, \langle a, a \rangle, \langle a, a, d \rangle, \langle a, a, e \rangle, \langle d \rangle, \langle e \rangle, \langle d, d \rangle, \langle d, e \rangle, \langle e, d \rangle$, $\langle e, e \rangle$. Note that every occurrence of b in a trace of Q ıı R is replaced by a; every occurrence of c is replaced *either* by *d* or by *e*.

In practice, it is useful to have events with multiple fields: for example, a description of a bank account might use events such as *deposit.50*, *deposit.100* and *withdraw.100*. Furthermore, a bank account should be prepared to accept deposits of any amount: the notation *deposit*?*x* represents a choice among all *deposit* events and the variable *x* becomes bound to the actual value of the second field. Thus, for example, the process *in*?*w* → *out.w* → STOP has among its traces $\langle in.13, out.13 \rangle$ and $\langle in.45, out.45 \rangle$.

Modelling the protocol's messages

For simplicity of presentation, let us consider a simplified version of the protocol in which *A* and *B* already know each other's public keys. Under these circumstances, *A* and *B* can forgo the communications with the key server *S*, resulting in a protocol in which only three messages need to be exchanged:

- A → B : {Na , A}Kb;
- B → A : {Na , Nb}Ka;

- A → B : {Nb}Kb.

$$\begin{aligned}&\text{INITIATOR}(a,n_a) =\\ &\text{Irunning.a?b} \rightarrow \text{comm.msg1.a.b.Encrypt.}K_b\text{.n}_a\text{.A}\\ &\rightarrow \text{comm.msg2.b.a.Encrypt.}K_a\ \text{?n?nb}\\ &\rightarrow \text{if not } (n == n_a)\\ &\text{then STOP}\\ &\text{else } (\textit{comm.msg3.a.b.Encrypt.}K_b\textit{.nb}\\ &\rightarrow \textit{Icommit.a.b}\\ &\rightarrow \textit{session.a.b} \rightarrow \text{STOP})\end{aligned}$$

Figure 8.4. The process Initiator($\mathbf{a}, \mathbf{n_a}$)

This simplification does not affect the final result of the analysis: the flaw uncovered also exists in the seven-message version of the protocol.

In the analysis that follows, three forms of compound events will represent these messages:

- msg1.A.B.Encrypt.Kb.Na.A;
- msg2.B.A.Encrypt.Ka .Na.Nb;
- msg3.A.B.Encrypt.Kb.Nb.

In each case, the event captures the message's role in the protocol, its sender and receiver, the key used to encrypt, and the contents of the encrypted message.

These events are further extended with prefixes (*comm*, *intercept*, *fake*) to represent messages that are communicated safely between *A* and *B*, intercepted by an intruder, or forged, respectively.

Modelling the agents

A CSP description for a generic initiator of this protocol ap-pears in Figure 8.4. Intuitively, the process INITIATOR(a, n_a) represents an initiator a who originally possesses the single nonce n_a and has public key K_a. The event *Irunning.a*?*b* represents *a*'s intent to initiate a run of the protocol with some agent *b*; the portion '?*b*' of this event indicates

that *b* can be instantiated to any valid principal of the system. (There is a similar event *Rrunning.b*?*a*' that represents a recipient *b*'s belief that it is engaged in a run of the protocol with a principal claiming to be *a*'.) The initiator's transmission of the first message in the protocol is represented by the event *comm.msg1.a.b.Encrypt.*K_b*.*n_a*.a*, where K_b is the public key associated with *b*; the initiator then waits for the response message from *b*, as represented by the event[2] *comm.msg2.b.a.Encrypt.*K_a*?n?nb*.

The initiator terminates the protocol if the nonce *n* received is not the nonce that it originally transmitted; this termination is represented by the deadlock process STOP.

If the nonce *does* match, then the initiator transmits the final message to *b* (per the event *comm.msg3.a.b.Encrypt.*K_b*.nb*) and commits to the protocol (represented by the event *Icommit.a.b*). The final event *session.a.b* abstracts the actual session between *a* and *b*.

As given, the process INITIATOR(*a*, n_a) does not incorporate the possibility of intercepted or faked messages. We can include these possibilities via a simple renaming, as in Figure 8.5, where we are able to fix a particular initiator *A* with nonce N_A. This renaming captures the notion that the Initiator's communications can be intercepted and that the messages it receives could be forged by an intruder, unbeknown to the Initiator (i.e. the Initiator behaves as if every communication is legitimate).

INIT = INITIATOR(A,N_A) [*comm.msg1* ← *comm.msg1*,
comm.msg1 ← intercept.msg1,
comm.msg2 ← comm.msg2,
comm.msg2 ← fake.msg2,
comm.msg3 ← comm.msg3,
comm.msg3 ← intercept.msg3]

Figure 8.5. The revised initiator Init

A responder process can be written in a similar fashion, after which the pair of agents are placed in parallel:

AGENTS = INIT [|*comm, session*|] RESP

Here, *comm* and *session* are used to constrain the behaviour of the processes: *comm* and *session* events may occur only if both agents participate in them simultaneously.

Modelling the intruder

In modelling the intruder, it pays to be as general as possible: if we encode particular types of attacks, then at best we can argue that the protocol is invulnerable to those specific attacks. Instead, we encode only the following general (and standard) assumptions about the intruder:

(1) the intruder can potentially overhear and intercept any message in the system;

(2) the intruder can decrypt any message encoded with her own public key;

(3) the intruder can replay intercepted messages (and alter any plaintext components of them), even if she cannot decrypt the message itself;

(4) the intruder can introduce new messages into the system, using any nonces she has available to her.

In particular, we can assume that the intruder *cannot* decrypt messages encrypted with keys that she does not possess. However, we can make no assumptions about the intruder's status within the network: the intruder may be an insider or an outsider.

The complete CSP description is fairly straightforward but rather long. As a simplification that illustrates the approach, Figure 8.6 provides a CSP description for an intruder in a system where the only messages being passed around are of the second type (i.e. those that contain two nonces encrypted); the true description of the intruder process also incorporates *comm*, *fake* and *intercept* events involving messages 1 and 3. The process INTR*(Ms, Ns)* is parameterized by two sets: *Ms*, which contains the undecrypted messages the intruder has intercepted so far, and *Ns*, the set of nonces she has collected so far. The *comm* events correspond to communications in which the intruder

is a legitimate participant (i.e. the intruder is the original sender or intended recipient). The *fake* events correspond to replays that the intruder performs: she can replay any message that she has already seen,[3] and she can modify the plaintext fields (i.e. the identities of senders and receivers) at will. Finally, the *intercept* events correspond to message interceptions: if the message is encoded with her own key, the intruder can decrypt it and add the two (now decrypted) nonces to her collection of known nonces; if not, then the message becomes one that she can replay at a later date.

$$
\begin{aligned}
&\text{INTR}(Ms, Ns) = \\
&\text{comm.msg2?b.a.Encrypt.k.n1.n2} \\
&\rightarrow \text{if } (\text{k} == \text{K}_\text{i}) \\
&\text{then INTR } (Ms, Ns \cup \{n1, n2\}) \\
&\text{else INTR } (\text{Ms} \cup \{\text{Encrypt.k.n1.n2}\}, \text{Ns}) \\
&\square\ \text{fake.msg2?a?b?m:Ms} \rightarrow \text{INTR(Ms, Ns)} \\
&\square\ \text{fake.msg2?a?b.Encrypt?k?n : Ns?n}^{-}\text{:Ns} \rightarrow \text{INTR(Ms, Ns)} \\
&\square\ \text{intercept.msg2?b.a.Encrypt.k.n1.n2} \\
&\rightarrow \text{if } (k == K_i) \\
&\text{then INTR}(Ms, Ns \cup \{n1, n2\}) \\
&\text{else INTR}(\text{Ms} \cup \{\text{Encrypt.k.n1.n2}\}, \text{Ns})
\end{aligned}
$$

Figure 8.6. The intruder process INTR($\mathbf{M_S}$, $\mathbf{N_S}$)

Finally, the entire system is defined by placing the agents and an intruder with nonce N_I in parallel:

SYSTEM=AGENTS[|$\boldsymbol{comm, fake, intercept}$|] INTR($\emptyset$, {$\boldsymbol{N_I}$})

Specifying and assessing authentication

Authentication requires that whenever a principal *A* thinks she has established a session with *B*, *B* has indeed been running the protocol with *A*. Because we wish to verify two-way authentication, there are two properties to specify:

(1) the initiator *A* commits to the session only if the responder *B* thinks he has participated in the protocol with *A*; and (2) the responder

B commits to the session only if the initiator *A* thinks she has participated in the protocol with *B*.

These two properties can be expressed in CSP as the processes *RA* (*receiver authentication*) and *IA* (*initiator authentication*):

$RA = Rrunning.A.B \rightarrow Icommit.A.B \rightarrow RA$

$IA = Irunning.A.B \rightarrow Rcommit.A.B \rightarrow IA$

To verify that the authentication properties hold, it suffices to perform the following two refinement checks:

$RA \; SYSTEM \setminus (\sum - \{Rrunning.A.B, Icommit.A.B\})$

$IA \; SYSTEM \setminus (\sum - \{Irunning.A,B, Rcommit.A.B\})$

That is, if every trace of *SYSTEM* (with all events other than *Rrunning.A.B* and *Icommit.A.B* hidden) is also a trace of *RA*, then the property *RA* is guaranteed to hold (and similarly for the property *IA*).

It turns out that the first refinement check succeeds, but the second check fails: there are traces of *SYSTEM* in which the event *Rcommit.A.B* occurs without a preceding *Irunning.A.B* event. For example, *SYSTEM* can perform the can the following sequence of events:

$Irunning.A.I$	(1)
$intercept.msg1.A.I.Encrypt.K_I.N_A.A$	(2)
$fake.msg1.A.B.Encrypt.K_B.N_A.A$	(3)
$Rrunning.A.B$	(4)
$intercept.msg2.B.A.Encrypt.K_A.N_A.N_B$	(5)
$fake.msg2.I.A.Encrypt.K_A.N_A.N_B$	(6)
$intercept.msg3.A.I.Encrypt.K_I.N_B$	(7)
$fake.msg3.A.B.Encrypt.K_B.N_B$	(8)
$Rcommit.A.B$	(9)

The existence of this trace highlights a potential ‘man in the middle’ attack, whereby the intruder participates as a legitimate recipient in one run of the protocol with A (the events in lines (1)–(2) and (6)–(7)) while impersonating A in a second run of the protocol with B (lines (3)–(5) and (8)–(9)). Ultimately, B commits to a session with A, despite the fact that A never even attempted to interact with B.

8.5 Associating keys and principals

Up until now, we have taken for granted the association between principals and keys. Correctly making this association is crucial, particularly when using a PKI. Associating keys with principals typically depends on two components.

(1) *Certification authorities:* principals who are recog-nized as having the authority to vouch for the correct-ness of the associations between keys and principals.

(2) *Certificates:* data structures that associate keys with principals. They are digitally signed by certification authorities to preserve the integrity of the cryptographic information.

Certificates are much like driver's licenses and certification authorities are like the network of authorities who issue those licenses. Chief among standards for public-key authentication services is the X.509 standard [8].

Figure 8.7 contains an example of a very simple multiple certification-authority (MCA) network, where principal *CA1* is the certification authority for principal *A*, *CA2* is the certification authority for principal *B*, and *CA1* has a certificate for CA2. In this example, using the notation of X.509, the following certificates exist.

- $CA1\langle\langle \boldsymbol{CA2} \rangle\rangle$: a certificate digitally signed with *CA1*'s private key certifying that key $CA2_p$ is *CA2*'s public key. The integrity of this certificate is checked using *CA1*'s public key $CA1_p$;
- $CA2\langle\langle \boldsymbol{B} \rangle\rangle$: a certificate digitally signed with *CA2*'s private key certifying that key B_p is *B*'s public key. The integrity of this certificate is checked using *CA2*'s public key $CA2_p$.

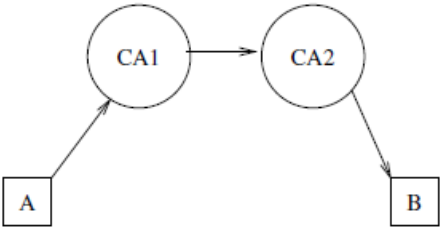


Figure 8.7. Multiple certification authorities

Informally, principal *A* can get principal *B*'s pudlic key using the above certeficates, provided that *A* knows *CAI*'s public key (i.e. $\boldsymbol{CAI_p}$). First, *A* uses $\boldsymbol{CAI_p}$ and the certeficate CAI $\langle\langle \boldsymbol{CA2} \rangle\rangle$ to get an integrity-chacked copy of *CA*2's pudlic key $\left(\boldsymbol{CA2_p}\right)$. In the notation of X.509, this step is represented by the equation

$$\boldsymbol{CA2_p = CAI_p \bullet CAI\langle\langle CA2\rangle\rangle}$$

where • denotes the operator that extracts a key from a certificate and checks its integrity using the supplied public key. Using $CA2_p$, *A* then extracts from the certificate $\boldsymbol{CA2\langle\langle B\rangle\rangle}$ an integrity-checked copy of B_p:

$$\boldsymbol{B_p = CA2_p \bullet CA2\langle\langle B\rangle\rangle}$$

Reasoning about certification

Lampson *et al.* [4] created a logic to reason about authentication in distributed systems. One of the main purposes of the logic is to answer questions such as 'Who is speaking?' and 'Whom does this key speak for or represent?'. While space considerations do not allow a complete description of the logic here, we give a flavour of the logic and its use by examining the MCA example with it.

There are two central notions necessary for our analysis.

(1) *Principals making statements.* This notion is denoted by *P says s*, where *P* is a principal and *s* is some (logical) statement. Implicit in *says* statements is the notion that the statement can be traced back to

some digitally signed statement. Principals can be people, machines, operating systems or cryptographic keys.

(2) *Principals speaking for principals.* The notion of principal P speaking for principal Q is denoted by $P \Rightarrow Q$. A common idiom is $P_p \Rightarrow P$, which indicates that P 's public key P_p speaks for P : statements that are digitally signed using P_p will be inferred to be statements made by P.

The logic is defined by approximately 19 axioms, including the following *handoff axiom*:

$(P \text{ says } (Q \Rightarrow P)) \supset (Q \Rightarrow P)$.

This handoff axiom says that whenever a principal P states that another principal Q speaks on P 's behalf, then Q does speak on P 's behalf. Related to this axiom is the property

$(P \Rightarrow Q) \supset ((P \text{ says } s) \supset (Q \text{ says } s))$.

This property—derivable from the logic's axioms—states that whenever a principal P speaks for Q and makes a statement s, it is safe to behave as if Q made the statement s.

Analysis of the MCA example

Returning to the example of Figure 8.7, it was shown how this logic helps us to reason about the certificates and the implicit trust assumptions that underlie A's trust in B_p as B's public key. That is, we are able to isolate sufficient assumptions under which A can conclude that $B_p \Rightarrow B$.

It turns out that the following five assumptions are sufficient for concluding $B_p \Rightarrow B$.

(1) $CA1_p \Rightarrow CA1$. A knows the public key of its certification authority (i.e. from A's perspective, the key $CA1_p$ speaks for $CA1$).

(2) $CA1_p \text{ says } (CA2_p \Rightarrow CA2)$. $CA1$'s public key $CA1_p$ is used to verify the validity of the certificate $CA1\langle\langle \boldsymbol{CA2} \rangle\rangle$.

(3) $CA2_p \text{ says } (B_p \Rightarrow B)$. Similarly, the key $CA2_p$ is used to verify the certificate $CA2\langle\langle \boldsymbol{B} \rangle\rangle$.

(4) $CA1 \Rightarrow CA2$. A trusts its certificate authority to speak for other

certificate authorities (*CA2* in this case).

(5) *CA2* ⇒ *B*. *A* knows that *B*'s certificate authority is *CA2*.

The proof of $\boldsymbol{B_p \Rightarrow B}$ from these assumptions is straightforward. From the derived property $(\boldsymbol{P \Rightarrow Q}) \supset \big((\mathbf{P\ says\ s}) \supset (\mathbf{Q\ says\ s})\big)$ and assumptions (1) and (2), we can deduce that *CAI says* $\big(\boldsymbol{CA2_p \Rightarrow CA2}\big)$.

Likewise, we can then use assumption (4) to deduce that $\boldsymbol{CA2\ says}\ \big(\boldsymbol{CA2_p \Rightarrow CA2}\big)$.

From the handoff axiom, we get that $\boldsymbol{CA2_p \Rightarrow CA2}$. At this point, the certificate for *B* in assumption (3) allow us to conclude that $\boldsymbol{CA2\ says}\ \big(\boldsymbol{B_p \Rightarrow B}\big)$. Another application of the derived property, this time using assumption (5), lets us deduce that $\boldsymbol{B\ says}\ \big(\boldsymbol{B_p \Rightarrow B}\big)$. Finally, the handoff axiom lets us conclude that $\big(\boldsymbol{B_p \Rightarrow B}\big)$, which was our original goal. The value of this analysis is twofold: (1) it makes explicit the trust assumptions being made, and (2) it assures a consistent treatment of certificates and assumptions about certification authorities.

8.6 Conclusions

In this chapter there is no attempt to give an exhaustive description of the applications of formal methods to the problem of assuring security: we have examined only a single session-based protocol, designed to provide mutual authentication using a PKI. In particular, we have ignored shared-key cryptographic systems as well as *store-and-forward* security (e.g., secure electronic mail).

Instead, we focused on a single protocol and sketched how multiple logical systems and formal models can provide insight into security issues from a variety of viewpoints. The choice of a particular formalism is driven in part by the properties one wishes to prove and also by the tools available. Both the BAN logic and the authentication of logic of are special-purpose modal logics, specifically designed to support reasoning about freshness and trust. They focus attention on what principals must be prepared to accept and to believe in order to trust in the correctness of a protocol. In contrast, the process algebra

CSP is a general-purpose language for describing and reasoning about the behaviour of concurrent systems. For this reason, it is well suited for reasoning about the high-level interactions and events that may occur during a run of a protocol.

These analyses demonstrate the subtlety of security properties and the importance of having rigorous methods for assessing the security of a system. Security properties are affected by timing and timeliness (e.g., present versus past runs of a protocol), trust (or lack thereof) in various principals and authorities, and cryptographic properties. Furthermore, there are nuances that arise between different levels of abstraction. The value of these formal methods is that they help in the detection of weaknesses and possible attacks, as well as making explicit any necessary assumptions that have been made.

Finally, the methods described in this chapter are all accessible to engineers with an understanding of predicate logic. The specialized logics can be embedded into theorem provers: [15] describes the embedding of a BAN-like logic into the HOL theorem prover [16]. Likewise, there are tools such as Lowe's Casper [17] that automatically translate abstract descriptions of security protocols into process-algebraic descriptions that can be analysed with model checkers.

Advancement questions

1. How can we use the formal methods to analyse protocol?
2. What is the BAN belief logic?
3. What does the security properties describe?
4. What do the key-distribution protocols enable?
5. What is the Needham–Schroeder protocol is designed for?
6. What do the CSP and CCS process algebras provide?
7. What the components the associating keys with principals depends on?
8. What are the main principals of the modelling the protocol's messages?

9. What are the main principals of the modelling the intruder?
10. What are the main principals of the specifying and assessing the authentication?

REFERENCES

[1] Susan Older Formal Methods for Assuring Security of Protocols / Susan Older, Shiu-Kai Chin // Computer Journal. – 2002. - Vol. 45, Issue 1. - Pp. 46-54.

[2] Needham, R. Using encryption for authentication in large networks of computers / Needham, R. and Schroeder, M. - Commun. ACM 1978, 21. - pp 993-999.

[3] Burroughs M. A Logic of Authentication / Burroughs M. Abadi M. and Needham R. // Report 39, Digital Equipment Corporation Systems Research Center. - Palo Alto, CA, February 1989.

[4] Lampson B. Authentication in distributed systems: theory and practice / Lampson B. Abadi, M. Burroughs M. and Wobber E.// ACM Trans. Comput. Syst., 10. 1992. - pp 265-310.

[5] Hoare C. A. R. Communicating Sequential Processes (.Series in Computer Science) / Hoare, C. A. R. - Prentice-Hall. London, 1985.

[6] Menzies A. J. Handbook of Applied Cryptography / Menzies A. J. von Oorschot P. C. and Vanstone S. A. - CRC Press, New York, 1996.

[7] Handbook of Applied Cryptography / Menzies, A. J., von Oorschot, P. C. and Vanstone S. A. - CRC Press, New York, 1996.

[8] Information Technology—Open Systems Interconnection—The Directory: Authentication Framework: ISO/EC 9594-8, 1995. - ISO Standart

[9] Gong L. Reasoning about belief in cryptographic protocols/ Gong L. Needham R. and Yahalom R. // In Proc. 1990 IEEEComput.

Soc. Symp. on Research in Security and Privacy, Oakland, CA, May. - 1990. - pp. 234-248.

[10] van Oorschot P. Extending cryptographic logics of belief to key agreement protocol / van Oorschot P. // In Proc. First ACM Conf on Computers and Communications Security, Fairfax, VA. - 1993 November. - pp. 232-243.

[11] Roscoe A. W. The Theory and Practice of Concurrency (Series in Computer Science) / A. W. Roscoe Prentice-Hall, London, 1998.

[12] Milner R. A Calculus of Communicating Systems/ R. Milner // Springer, Berlin. -1980 . - Vol. 92.

[13] Formal Systems (Europe) Ltd / Failures-Divergence Refinement: EDR2 User Manual. - Oxford, 1997.

[14] Lowe G. An attack on the Needham-Schroeder public-key authentication protocol / G. Lowe // Inform. Process. Lett., 56, 1995. - pp 131-133.

[15] Lowe G. Breaking and fixing the Needham- Schroeder public-key protocol using fdr / G. Lowe. - Software Concepts Tools, 17, 1996. - pp 93-102.

[16] 15 Schubert T. A mechanized logic for secure key escrow protocol verification/ T. Schubert and S. Mocas // In Proc. Eighth Int. Workshop on Higher Order Logic Theorem Proving and Its Applications. - 1995.

[17] 16 Gordon M. J. C. Introduction to HOL: A Theorem Proving Environment for Higher Order Logic / M. J. C. Gordon and T. F. Melham. - Cambridge University Press, Cambridge, 1993.

[18] 17 Lowe G. Casper: a compiler for the analysis of security protocols / G. Lowe // J. Comput. Security, 6, 1998. - pp 53-84.

CHAPTER 9. FORMAL METHODS FOR THE ANALYSIS OF SECURITY PROTOCOLS

Content of the CHAPTER 9

Soundness of Formal Encryption

Relating Symbolic and Computational cryptography has attracted the interest of the research community. Several different directions have been taken to bridge the gap between the two models: some extend the existing results by including more primitives; some by adapting existing results from the passive adversary scenario to the active adversary scenario; some others by including new primitives from computational cryptography.

This chapter is one more effort to bridge the gap between these two communities. Mainly, let us try to bridge two gaps that exist since the early results of Abadi and Rogaway. The first is the non-existence of soundness results in the presence of key-cycles. Key-cycles do not present a problem from the symbolic point of view. One may even argue that protocols that create messages with encryption cycles may be avoided and are just result of bad engineering. But, even if our protocols are restricted to the cases where no cycles are created, no one can ensure us that an adversary is not able to create cyclic encryptions and that these would not cause problems. Studying this is part of the work in this chapter. We can show that it is possible to close this gap but for that the use of new definitions of security is needed.

The second gap is to be closen is to extend the original Abadi and Rogaway result when the encryption scheme used provides less security guarantees. The encryption scheme used in their original result is very strong and arguably impossible to realise in many contexts. The aim is to relax such conditions by allowing the use of weaker encryption schemes but still achieving similar soundness results. It will allow encryption schemes that reveal the length of the encrypted plaintext. Let us study this particular example and then show a uniform

framework with which it will be able to characterise a large family of encryption schemes.

9.1 The Abadi-Rogaway Soundness Theorem

The main definitions and results of Abadi and Rogaway's original work are briefly summarised [4, 2]. In particular, let us start presenting the formal model, then describe the computational model, and then introduce the notion of soundness. Furthermore, let us introduce the notion of completeness, which can be viewed as the counter-point to soundness.

The Formal Model

In this model, messages (or *expressions*) are defined at a very high level of abstraction. The simplest expressions are symbols for atomic keys and bit-strings. More complex expressions are created from simpler ones via encryption and concatenation, which are defined as abstract, 'black-box' constructors.

Definition 9.1 (Symmetric Expressions). Let Keys = $\{\mathbf{K_1}; \mathbf{K_2}; \mathbf{K_3}; ...\}$ be an infinite discrete set of symbols, called the set of symmetric keys. Let Blocks be a finite subset of $\{\mathbf{0}, \mathbf{1}\}^*$. We define the *set of expressions*, Exp, by the grammar:

Exp ::= Keys | Blocks | (Exp*;* Exp) | $\{\mathbf{Exp}\}_{\mathbf{Keys}}$

Let Enc ::= $\{\mathbf{Exp}\}_{\mathbf{Keys}}$. We will denote by *Keys*(*M*) the set of all keys occurring in *M*. Expressions of the form $\{\boldsymbol{M}\}_{\boldsymbol{K}}$ are called *encryption terms*.

Expressions may represent either a single message sent during an execution of the protocol, or the entire knowledge available to the adversary. In this second case, the expression contains not only the messages sent so far, but also any additional knowledge in the adversary's possession.

Let us define two formal expressions are indistinguishable to the adversary. Intuitively, this occurs when the only differences between the two messages lie within encryption terms that the adversary cannot decrypt. In order to rigorously define this notion, we first need to

formalise when an encryption term is 'undecryptable' by the adversary, which in turn requires us to define the set of keys that the adversary can learn from an expression.

An expression might contain keys in the clear. The adversary will learn these keys, and can then use them to decrypt encryption terms of the expression—which might reveal yet more keys. By repeating this process, the adversary can learn the set of *recoverable decryption keys*:

Definition 9.2 (Subexpressions, Visible Subexpressions, Recoverable Keys, Undecryptable Terms, B-Keys). We define the *set of subexpressions* of an expression *M*, *sub* (*M*), as the small-est subset of expressions containing *M* such that:

- $(M_1; M_2) \in sub\ (M) \Rightarrow M_1 \in sub\ (M)$ and $M_2 \in sub\ (M)$, and $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in$ sub (M) $\Rightarrow \boldsymbol{M'} \in$ sub (M).

We say that *N* is a subexpression of *M*, and denote it by *N M*, if $N \in sub\ (M)$.

The set of *visible subexpressions* of a symmetric expression *M*, *vis* (*M*), is the smallest subset of expressions containing *M* such that:

- $(M_1; M_2) \in vis\ (M) \Rightarrow M_1 \in vis\ (M)$ and $M_2 \in vis\ (M)$, and $\{\boldsymbol{M'}\}_{\boldsymbol{K}}$ and K $\in$ vis (M) $\Rightarrow \boldsymbol{M'} \in$ vis (M).

The *recoverable keys of* a (symmetric) expression *M*, *R-Keys*(*M*), are those that an adversary can recover by looking at an expression. That is, *R-Keys*(*M*) = *vis* (*M*) $\cap$ *Keys*(*M*).

We say that an encryption term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in vis\ (M)$ is *undecryptable* in *M* if $K \notin$ *R-Keys*(*M*). Among the non-recoverable keys of an expression *M*, there is an important subset denoted by *B-Keys*(*M*). The set *B-Keys*(*M*) contains those keys which encrypt the outermost undecryptable terms. Formally, for an expression *M*, we define *B-Keys*(*M*) as

$$B\text{-}Keys(M)=\{\boldsymbol{K} \in \boldsymbol{Keys(M)} \mid \{\boldsymbol{M}\}_{\boldsymbol{K}} \in \boldsymbol{vis}\ (\boldsymbol{M}) \textbf{ but } \boldsymbol{K} \notin \boldsymbol{R-Keys(M)g}\ \}$$

Example 9.1. Let *M* be the following expression

$$((\{\boldsymbol{Q}\}_{\boldsymbol{K_6}}, \{\{\boldsymbol{K_7}\}_{\boldsymbol{K_1}}\}_{\boldsymbol{K_4}} (\boldsymbol{K_2}, \{(\{\boldsymbol{001}\}_{\boldsymbol{K_3}}, \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}})\}_{\boldsymbol{K_5}}), \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}})).$$

In this case, *Keys*(*M*) = $\{\boldsymbol{K_1}; \boldsymbol{K_2}; \boldsymbol{K_3}; \boldsymbol{K_4}; \boldsymbol{K_5}; \boldsymbol{K_6}; \boldsymbol{K_7}\}$. The set of recoverable keys of *M* is *R-Keys*(*M*) =$\{\boldsymbol{K_2}; \boldsymbol{K_5}; \boldsymbol{K_6}\}$, because an adversary sees the non-encrypted K_2, and with that he can decrypt $\{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}$, hence recovering K_5; then, decrypting twice with K_5, K_6 can be revealed. We also have that *B-Keys*(*M*) = $\{\boldsymbol{K_3}; \boldsymbol{K_4}\}$.

The formal model allows expressions to contain *key cycles*:

Definition 9.3 (Key-Cycles). An expression *M* contains a *key-cycle* if it contains encryption terms $\{\boldsymbol{M_1}\}_{\boldsymbol{K_1}}, \{\boldsymbol{M_2}\}_{\boldsymbol{K_2}}, \ldots, \{\boldsymbol{M_n}\}_{\boldsymbol{K_n}}$ (where $\{\boldsymbol{M_i}\}_{\boldsymbol{K_i}}$denotes the encryption of the message M_i with the key K_i) and K_{i+1} *v* M_i and K_1 *v* M_n. In this case we say that we have a key-cycle of length *n*.

According to our definition, expressions such as $\{\{\boldsymbol{M}\}_{\boldsymbol{K}}\}_{\boldsymbol{K}}$ are not considered cyclic. The original result of Abadi and Rogaway does not apply to expressions with key cycles. The aim is to correct this weaknes.

The AR Equivalence of Formal Expressions

A visible encryption term will appear ‘opaque’ to the adversary if and only if it is protected by at least one non-recoverable decryption key. Thus, we can to say that two expressions are equivalent if they differ only in the contents of their ‘opaque’ encryption terms. To express this, Abadi and Rogaway define the *pattern* of an expression through which equivalence of expressions will be obtained:

Definition 9.4 (Pattern (Classical)). We define the *set of patterns*, Pat, by the grammar:

Pat ::= Keys | Blocks | (Pat*;* Pat) | $\{\mathbf{Pat}\}_{\mathbf{Keys}}$

The pattern of an expression *M*, denoted by *pattern*(*M*), is derived from *M* by replacing each encryption term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in$ *vis* (*M*) (where $K \notin$ = *R-Keys*(*M*)) by for two patterns *P* and *Q*, *P* = *Q* is defined the following way:

- if $P \in$ Blocks ∪ Keys, then *P* = *Q* iff *P* and *Q* are identical;
- if *P* is of the form, then *P* = *Q* iff *Q* is of the form;

- if P is of the form (P_1; P_2), then $P = Q$ iff Q is of the form (Q_1; Q_2) where $P_1 = Q_1$ and $P_2 = Q_2$;
- if P is of the form $\{\boldsymbol{P'}\}_{\boldsymbol{K}}$, then $P = Q$ iff Q is of the form $\{\boldsymbol{Q'}\}_{\boldsymbol{K}}$where P'=Q'.

(Note that we call these 'classical' patterns. This is to distinguish them from the more complex patterns that we will consider later.)

Two expressions are equivalent if their patterns are equal. However, consider two very simple formal expressions K_1 and K_2. Then these formal expressions would not be equivalent. On the other hand, these two expressions have the same meaning: a randomly drawn key. Despite being given different names, they both represent samples from the same distribution. It does not matter if we replace one of them with the other. More generally, we are able to formalise the notion of equivalence in such a way that renaming the keys yields in equivalent expression. Therefore, two formal expressions should be equivalent if their patterns differ only in the names of their keys.

Definition 9.5 (Key-Renaming Function). A bijection $\boldsymbol{\sigma}$: Keys → Keys is called a *key-renaming function*. For any expression (or pattern) M, M $\boldsymbol{\sigma}$ denotes the expression (or pattern) obtained from M by replacing all occurrences of keys K in M by $\boldsymbol{\sigma}$ (K).

We are finally able to formalise the symbolic notion of equivalence:

Definition 9.6 (Equivalence of Expressions). We say that two expressions M and N are *equiv- alent*, denoted by $M \cong N$, if there exists a key-renaming function $\boldsymbol{\sigma}$ such that *pattern*(M) = *pattern*(N $\boldsymbol{\sigma}$).

The Computational Model

The fundamental objects of the computational world are strings, strings = $\{\mathbf{0}, \mathbf{1}\}^*$, and families of probability distributions over strings. These families are indexed by a *security parameter* $\eta \in$ parameters = $\mathbb{N}$ (which can be roughly understood as key-lengths). Two distribution families $\{\boldsymbol{D}_{\boldsymbol{\eta}}\}_{\boldsymbol{\eta}\in\mathbb{N}}$ and $\{\boldsymbol{D}'_{\boldsymbol{\eta}}\}_{\boldsymbol{\eta}\in\mathbb{N}}$ are *indistinguishable* if no efficient algorithm can determine from which distribution a value was sampled:

Definition 9.7 (Negligible Function). A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is said to be *negligible*, written

$f(n) \leq \text{neg}(n)$, if for any $c > 0$ there is an $n_c \in \mathbb{N}$ such that $f(n) \leq \boldsymbol{n^{c-}}$whenever $n \geq n_c$.

Definition 9.8 (Indistinguishability). Two families $\{\boldsymbol{D_\eta}\}_{\boldsymbol{\eta}\in\mathbb{N}}$ and $\{\boldsymbol{D'_\eta}\}_{\boldsymbol{\eta}\in\mathbb{N}}$, are *indistinguish-able*, written $\boldsymbol{D_\eta \approx D'_\eta}$, if for all PPT adversaries A,

$$\left|\boldsymbol{Pr}\left[\boldsymbol{d} \leftarrow \boldsymbol{D_\eta}; \boldsymbol{A}(\boldsymbol{1^\eta}, \boldsymbol{d}) = \boldsymbol{1}\right] - \boldsymbol{Pr}\left[\boldsymbol{d} \leftarrow \boldsymbol{D'_\eta}; \boldsymbol{A}(\boldsymbol{1^\eta}, \boldsymbol{d}) = \boldsymbol{1}\right]\right| \leq \boldsymbol{neg}(\boldsymbol{\eta})$$

In this model, pairing is an injective pairing function [.,.] : strings × strings → strings such that the length of the result only depends on the length of the paired strings. An encryption scheme is a triple of algorithms (*K; E; D*) with key generation *K*, encryption *E* and decryption *D*. Let plaintexts, ciphertexts, and keys be nonempty subsets of strings. The set coins is some probability field that stands for coin-tossing, *i.e.*, randomness.

Definition 9.9 (Symmetric Encryption Scheme). A *computational symmetric encryption scheme* is a triple $\prod$ = (*K; E; D*) where

- *K* : parameters × coins → keys is a key-generation algorithm;
- *E* : keys × strings × coins → ciphertexts is an encryption function;
- *D* : keys × strings → plaintexts is such that for all *k* ∈ keys and $\boldsymbol{\omega} \in$ coins,

D(K,E(k,m,w))=m for all m ∈ plaintexts

D(K,E(k,m',w))=⊥ for all m' ∉ plaintexts

All of *K*, *E* and *D* are computable in polynomial-time in the length of the security parameter. This definition, note, does not include any notion of security, and this must be defined separately. In fact, there are several different such definitions. Abadi and Rogaway, in their work, consider a spectrum of notions of their own devising, from 'type-0' to 'type-7.' Their main result uses the strongest of these notions, type-0.

Definition 9.10 (Type-0 Security). We say that a computational encryption scheme is type-0 secure if no probabilistic polynomial-time adversary A can distinguish the pair of oracles ($E(k;\ \cdot\ ); E(k';\ \cdot\ )$) from the pair of oracles ($E(k;\ 0); E(k;\ 0)$) as k and k' are randomly generated. That is, for any probabilistic polynomial-time algorithm, A,

$$\boldsymbol{Pr}\left[\boldsymbol{k}, \boldsymbol{k'} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}): \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\cdot),\boldsymbol{E}(\boldsymbol{k'},\cdot)}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] - \boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}): \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},0),\boldsymbol{E}(\boldsymbol{k},\cdot 0)}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] \leq \boldsymbol{neg}(\boldsymbol{\eta})$$

Intuitively the above formula says the following: The adversary is given one of two pairs of oracles, either ($E(k;\ \cdot); E(k';\ \cdot)$) or ($E(k;\ 0); E(k;\ 0)$) (where the keys were randomly generated prior to handing the pair to the adversary), but it does not know which. Then, the adversary can perform any (probabilistic polynomial-time) computation, including several queries to the oracles. It can even query the oracles with messages that depend on previously given answers of the oracles. (The keys used by the oracles for encryption do not change while the adversary queries the oracles.) After this game, the adversary has to decide with which pair of oracles it was interacting. The adversary wins the game if he can decide for the correct one with a probability bigger than $\frac{\mathbf{1}}{\mathbf{2}}$, or (equivalently) if it can distinguish between the two. If this difference is negligible, as a function of $\boldsymbol{\eta}$, we say the encryption scheme is type-0 secure.

As Abadi and Rogaway show, type-0 security is strong enough to provide *soundness* to the formal model. But to see this, we must first explain how the two models can be related.

The Interpretation Function, Soundness and Completeness

In order to prove any relationship between the formal and computational worlds, we need to define the *interpretation* of expressions and patterns. Once an encryption scheme is picked, we can define the interpretation function Φ, which assigns to each expression or pattern M a family of random variables $\{\boldsymbol{\Phi}_{\boldsymbol{\eta}}(\boldsymbol{M})\}_{\boldsymbol{\eta}\in\mathbb{N}}$ such that each

$\Phi_\eta(M)$ takes values in strings. As in Abadi and Rogaway [2], this interpretation is defined in an algorithmic way. Intuitively,

- blocks are interpreted as strings;
- each key is interpreted by running the key generation algorithm;
- pairs are translated into computational pairs;
- formal encryptions terms are interpreted by running the encryption algorithm on the inter-pretation of the plaintext and the interpretation of the key.

For an expression *M*, we will denote by $[\![M]\!]_{\Phi_\eta}$ the distribution of $\Phi_\eta(M)$ and by $[\![M]\!]_\Phi$ the ensemble of $\left\{[\![M]\!]_{\Phi_\eta}\right\}_{\eta\in\mathbb{N}}$.

Then soundness and completeness are defined in the following way:

Definition 9.11 (Soundness (Classical)). We say that an interpretation is *sound in the classical sense*, or that an encryption scheme *provides classical soundness*, if the interpretation Φ (result-ing from the encryption scheme) is such that for any given pairs of expressions *M* and *N*

$$M \cong N \Rightarrow [\![M]\!]_\Phi \approx [\![N]\!]_\Phi$$

The primary result of Abadi and Rogaway given in [2] is that type-0 security provides classical soundness if the expressions *M* and *N* have no key-cycles.

Soundness has a counterpart, completeness. One can consider soundness to be the property that formal indistinguishability always becomes computational indistinguishability. One can think of completeness as the converse: computational indistinguishability is always the result of formal indistinguishability:

Definition 9.12 (Completeness (Classical)). We say that an interpretation is *complete* (in the classical sense), or that an encryption scheme *provides (classical) completeness*, if the interpre-tation Φ (resulting from the encryption scheme) is such that

$$[\![M]\!]_\Phi \approx [\![N]\!]_\Phi \Rightarrow M \cong N$$

for any expressions *M* and *N*.

We remark that for the proofs of the soundness and completeness results, it was convenient for Abadi and Rogaway to introduce the interpretation of any pattern M (although this is not absolutely necessary). Therefore, boxes are interpreted as well, such that □ is interpreted by running the encryption algorithm on the fixed plaintext 0 and a ran-domly generated key.

The precise definition of $\boldsymbol{\Phi_{\eta}(M)}$ for any pattern M is given by the algorithms in Figure 9.1. We are able to note that these algorithms are fully defined for patterns, and because the grammar for patterns contains the grammar for expressions as a sub-grammar, they are fully defined for expressions as well.

algorithm INITIALIZE($\mathbf{1^{\eta}}$; M)
for $K \in Keys(M)$ do $\boldsymbol{\tau}(K) \leftarrow K(\mathbf{1^{\eta}})$
let $\mathbf{k_0} \leftarrow K(\mathbf{1^{\eta}})$
algorithm CONVERT(M)
if $M = K$ where $K \in$ Keys then return $\boldsymbol{\tau}$ (K)
if $M = B$ where $B \in$ Blocks then return B
if $M = (M_1; M_2)$ then
$\mathbf{x} \leftarrow \text{CONVERT}(M_1)$
$\mathbf{y} \leftarrow \text{CONVERT}(M_2)$ return $[x; y]$
if $M = \{\mathbf{M_1}\}_{\mathbf{K}}$ then
$\mathbf{x} \leftarrow \text{CONVERT}(M_1)$ $\mathbf{y} \leftarrow \mathbf{E}(\boldsymbol{\tau}(\mathbf{K}), \mathbf{x})$
return y
if $M =$ □, then $\mathbf{y} \leftarrow \mathbf{E}(\mathbf{k_0}, \mathbf{0})$
return y

Figure 9.1: Algorithmic components of the interpretation function

9.2 Soundness in the Presence of Key-Cycles

Key-cycles do not cause a problem with completeness, however, one of the weaknesses of the original Abadi-Rogaway's result is that it is not possible to prove soundness for expressions that included key-cycles. So let us address this problem in this section starting by showing that, soundness in the presence of key-cycles is not possible to

prove with the security notion adopted by Abadi and Rogaway. Let us consier a new notion of security, KDM-security as a solution for the problem. In order to prove soundness, it also needed to extend our formal model, and after that show that with this new definition of security it is possible to obtain soundness even in the presence of key-cycles.

Type-0 Security is Not Enough

In this section let us show that type-0 security is not strong enough to ensure soundness in the case of key-cycles. That is, let us demonstrate that it is possible to construct encryption schemes that are type-0, but fail to provide soundness in the presence of key-cycles.

Theorem 9.1. *Type-0 security does not imply soundness. That is, if there exists an encryption scheme that is type-0 secure, then there exists another encryption scheme which is also type-0 secure but does not provide soundness.*

Proof. This is shown via a simple counter-example. Assuming that there exists a type-0 se-cure encryption scheme, we will use it to construct another scheme which is also type-0 secure. However, we will show that this new scheme allows the adversary to distinguish one particular expression M from another particular expression N, even though M ≅ N.

Let *M* be $\{\boldsymbol{K}\}_{\boldsymbol{K}}$ and let *N* be the expression $\{\boldsymbol{K_1}\}_{\boldsymbol{K_2}}$. Since these two expressions are equiv-alent, an encryption scheme that enforces soundness requires that the family of distributions:

$$\{\boldsymbol{k} \leftarrow \mathbf{K}(\mathbf{1^{\eta}}); \mathbf{c} \leftarrow \mathbf{E}(\mathbf{k}, \mathbf{k}) : \mathbf{c}\}_{\boldsymbol{\eta} \in \mathbb{N}}$$

be indistinguishable from the family of distributions:

$$\{\boldsymbol{k_1} \leftarrow \mathbf{K}(\mathbf{1^{\eta}}); \boldsymbol{k_2} \leftarrow \mathbf{K}(\mathbf{1^{\eta}}); \mathbf{c} \leftarrow \mathbf{E}(\mathbf{k_1}, \mathbf{k_2}) : \mathbf{c}\}_{\boldsymbol{\eta} \in \mathbb{N}}$$

However, this is not implied by Definition 2.10. Let ∏ = (*K; E; D*) be a type-0 secure encryption scheme. We assume that ¦ is such that keys and ciphertexts have different formats. Then, using ¦, we can construct a second type-0 secure encryption scheme ∏ ' = (K', E', D') as follows:

- Let $K' = K$,
- Let E' be the following algorithm:

$$E'(k,m) = \begin{cases} k & if\ m = k \\ E(k,k) & if\ E(k,m) = k \\ E(k,m) & otherwise \end{cases}$$

- Let D' be the following algorithm:

$$D'(k,c) = \begin{cases} k & if\ c = k \\ D(k,k) & if\ c = E(k,k) \\ D(k,c) & otherwise \end{cases}$$

The scheme ∏' acts exactly like ∏ unless the encryption algorithm E' is called on a pair (*k; k*). It is easy to see that this scheme is also type-0 secure.

To see this, suppose that ∏' is not type-0 secure. That is, there exists some adversary A which can distinguish the pair of oracles (E' (k, •), E' (k', •)) from the pair (E' (k, 0), E' (k, 0)). There are two possibilities. Suppose that the adversary queried the oracle on *k* or *k'*. Then it would certainly be able to distinguish the oracle-pairs, but this also means that the adversary can produce the secret symmetric key to the scheme ∏. Thus, the encryption scheme ∏ cannot be secure in any sense, much less type-0. Suppose, on the other hand, the adversary did not query the oracles on *k* or k' but managed to distinguish between the oracle pairs anyway. Then it was able to do so even though the encryption scheme ∏' acted exactly like ∏, and so ∏ cannot be type-0 secure.

Thus, the new scheme ∏' must also be type-0 secure. However, it does not guarantee in-distinguishability for the two distributions above. The first distribution will output always the encryption key while the second outputs a ciphertext, and these two distributions are easily distinguished by form alone.

Remark 9.1. We note that in the proof, the expression *M* contains a key-cycle of length 1. What if all key-cycles are of length 2 or more? This question remains open. That is, there is no known type-0 secure encryption scheme which fails to provide soundness for key-cycles that are of length two or more.

Because type-0 encryption implies types 1 through 7, Theorem 9.1 implies that soundness with key-cycles cannot be provided by the security definitions devised by Abadi and Rogaway. In the next section, shows that the soundness property can be met with *new* computational definitions.

KDM-Security

In the last section, it was shown that the notions of security found in [4, 2] are not strong enough to enforce soundness in the presence of key-cycles. However, *key-dependent message* (KDM) security, which was introduced by Black *et al.* [3] (and in a weaker form by Camenisch and Lysyanskaya [5]), is strong enough to enforce soundness even in this case.

KDM security both strengthens and weakens type-0 security. Recall that type-0 security allows the adversary to submit messages to an oracle which does one of two things:

- it could encrypt the message twice, under two different keys, or it could encrypt the bit 0 twice, under the same key.

An encryption scheme is type-0 secure if no adversary can tell which of these is being done. For KDM security, however, the game is slightly different. To over-simplify:

- the oracle in the KDM-security encrypts once, under only one key;
- further, it encrypts either the message, or a *string* of 0's of equivalent length;
- however, it is willing to encrypt not just messages from the adversary, but also (more generally) *functions of the secret key*.

The first two of these differences make KDM security weaker than type-0 security. Specifically type-0 security conceals both the length of the plaintext and whether two ciphertext were created using the same encryption key or different ones. KDM security does not necessarily conceal either of these things. The last difference, however, is a significant strengthening. As its name suggests, KDM security remains

strong even when the messages depend on the secret key— which, as Theorem 2.1 shows, is not necessarily true for type-0 security.

To provide the full picture, KDM security is defined in terms of *vectors* of keys and functions over these vectors. It is also defined in terms of oracles $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$ and $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$, which work as follows:

- suppose that for a fixed security parameter $\eta \in \mathbb{N}$, a vector of keys is given $\bar{\boldsymbol{k}} = \{\boldsymbol{k_i} \leftarrow \mathbf{K}(\mathbf{1}^{\boldsymbol{\eta}})\}_{\boldsymbol{i}\in\mathbb{N}}$ (In each run of the key-generation algorithm independent coins are used.) The adversary can now query the oracles providing them with a pair (j, g), where $j \in \mathbb{N}$ and $g : \boldsymbol{keys}^{\infty} \rightarrow \{\mathbf{0},\mathbf{1}\}^{*}$ is a constant length, deterministic function:

96. - The oracle $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$ when receiving this input returns $c \leftarrow \boldsymbol{E}\left(\boldsymbol{k_j}, \boldsymbol{g}(\bar{\boldsymbol{k}})\right)$.
97. - The oracle $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$ when receiving this same input returns $c \leftarrow \boldsymbol{E}\left(\boldsymbol{k_j}, \mathbf{0}^{|\boldsymbol{g}(\bar{\boldsymbol{k}})|}\right)$.

The challenge facing the adversary is to decide whether it has interacted with oracle $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$ or oracle $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$. Formally:

Definition 9.13 (Symmetric-KDM Security). Let $\prod = (K,E,D)$ be a symmetric encryption scheme. Let the two oracles $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$ and $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$ be as defined above. We say that the encryption scheme is *(symmetric) KDM-secure* if for all PPT adversaries A:

$$\mathbf{Pr}\left[\bar{\boldsymbol{k}} \leftarrow \mathbf{K}(\mathbf{1}^{\boldsymbol{\eta}}): \mathbf{A}^{\boldsymbol{Real}_{\bar{\boldsymbol{k}}}}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] - \mathbf{Pr}\left[\bar{\boldsymbol{k}} \leftarrow \mathbf{K}(\mathbf{1}^{\boldsymbol{\eta}}): \mathbf{A}^{\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] \leq \boldsymbol{neg}(\boldsymbol{\eta})$$

Remark 9.2. We note that although all known implementations of KDM-security are in the random-oracle model, this definition is well-founded even in the standard model. We also note that this definition is phrased in terms of indistinguishability. One could also imagine analogous defini-tions phrased in terms of non-malleability, but an exploration of those are beyond the scope of this dissertation.

We note that KDM-security implies type-3 security:

Definition 9.14 (Type-3 Security). Let $\prod = (K,E,D)$ be a symmetric encryption scheme. We say that the encryption-scheme is *type-3 secure*

if no PPT adversary A can distinguish the oracles $\boldsymbol{E(k,\bullet)}$ and $\boldsymbol{E(k;\ 0^{|\bullet|})}$ as *k* is randomly generated, that is, for all PPT adversaries A:

$$\boldsymbol{Pr\left[k \leftarrow \mathrm{K}(1^{\eta}): \mathrm{A}^{E(k,\bullet)}\right] - Pr\left[k \leftarrow \mathrm{K}(1^{\eta}): \mathrm{A}^{E(k;\,0^{|\bullet|})}(1^{\eta}) = 1\right] \leq neg(\eta)}$$

In fact, the definition of type-3 encryption is exactly the same as that for KDM-security, except that the adversary must submit concrete messages to the encryption oracle instead of functions. But since the functions submitted in KDM security can be constant function that always produce a single output, the type-3 security 'game' is a special case of that for KDM security.

On the other hand, KDM security does not attempt to conceal the length of the plaintext (type-1 security) or that two ciphertexts were created with the same key (type-2 security). It will be impossible, therefore, for KDM security to provide soundness in the classical sense (Defini-tion 2.11). Nonetheless, a weaker form of soundness can be achieved if the formal model is also weakened slightly.

A New Formal Model

In this section, let us consider a weaker version of the formal model—one that allows formal en-cryption to leak partial information about the plaintext and key. Here let us focus on the partial leakage allowed (in the computational model) by KDM security: the length of the plaintext, and whether two different ciphertexts were created using the same key.

To model the leakage of plaintext length, we first need to add the very concept of 'length' to the formal model.

Definition 9.15 (Formal Length). A formal length-function is a function symbol with fresh letter *l* satisfying at least the following identities:

- for all blocks B_1 and B_2, $l(B_1) = l(B_2)$ iff $|\boldsymbol{B_1}| = |\boldsymbol{B_2}|$;
- for all expression *M* and key-renaming function σ, $l(M) = l(M\,\sigma)$
- if $l(M_1) = l(N_1)$, $l(M_2) = l(N_2)$ then $l((M_1; M_2)) = l((N_1; N_2))$,

and

- if $l(M) = l(N)$, then for all K_i, $l(\{\boldsymbol{M}\}_{\boldsymbol{K_i}}) = l(\{\boldsymbol{N}\}_{\boldsymbol{K_i}})$.

These are the identities that a formal length function *mini-mally* has to satisfy. There may be more. In fact, if only these properties are assumed, there is no hope to obtain completeness. It follows that for any key-renaming function σ, and expression M, $l(M) = l(M\sigma)$.

Given this, it is straightforward to add the required leakage to the formal model. If patterns represents those aspects of an expression that can be learned by the adversary, then patterns must now reveal the plaintext-length and key-names for undecryptable terms:

Definition 9.16 (Pattern (Type-3)). We define the *set of patterns*, Pat, by the grammar:

Pat ::= Keys | Blocks | (Pat; Pat) | $\{\boldsymbol{Pat}\}_{\boldsymbol{Keys}}$ | $\square_{\boldsymbol{Keys},\boldsymbol{l}(\mathbf{Exp})}$

The type-3 pattern of an expression M, denoted by $pattern_3(M)$, is derived from M by replacing each encryption term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in \boldsymbol{vis}(\boldsymbol{M})\big(\boldsymbol{where}\ \boldsymbol{K} \notin \boldsymbol{R-Keys}(\boldsymbol{M})\big)\boldsymbol{by}\ \square_{\boldsymbol{K},\boldsymbol{l}(\mathbf{M'})}$.

Note that the only difference between a type-3 pattern and a classical pattern is that an unde-cryptable term $\{\boldsymbol{M}\}_{\boldsymbol{K}}$becomes $\square_{\boldsymbol{K},\boldsymbol{l}(\mathbf{M})}$ (*i.e.* labelled with the key and length) in type-3 patterns instead of merely $\square$ in classical patterns.

Our notion of formal equality must be updated as well. For two patterns P and Q, $\boldsymbol{P} \cong_{\mathbf{3}} \boldsymbol{Q}$ is defined the following way:

Definition 9.17 (Formal Equivalence (Type-3)). We first introduce the relation $=_3$ between patterns:

- if $P \in$ Blocks $\cup$ Keys, then $P =_3 Q$ iff P and Q are identical;
- if P is of the form $\square_{\boldsymbol{K},\boldsymbol{l}(\mathbf{M'})}$, then $P =_3 Q$ iff Q is of the form $\square_{\boldsymbol{K},\boldsymbol{l}(\mathbf{N'})}$, and $l(M') = l(N')$ in the sense of Definition 2.15;
- if P is of the form $(P_1; P_2)$, then $P =_3 Q$ iff Q is of the form $(Q_1; Q_2)$ where $P_1 =_3 Q_1$ and $P_2 =_3 Q_2$.;
- if P is of the form $\{\boldsymbol{P'}\}_{\boldsymbol{K}}$, then $P =_3 Q$ iff Q is of the form $\{\boldsymbol{Q'}\}_{\boldsymbol{K}}$, where $P' =_3 Q'$.

With this, we say that expressions M and N are *equivalent in the type-3 sense (writen* $\boldsymbol{M \cong_3 N}$*)* if there exists a key-renaming function σ such that *pattern*$_3$ (M) =$_3$ *pattern*$_3$ (Nσ). (Since a key-renaming function replaces all occurrences of K with σ(K), we note σ, $\square_{\boldsymbol{K},\boldsymbol{l}(\mathbf{M})}$ will become $_{\boldsymbol{\sigma}(\mathbf{K}),\boldsymbol{l}(\boldsymbol{M}\boldsymbol{\sigma})}\bullet$)

Lastly, the above change to formal equivalence requires that the notions of soundness and completeness be similarly altered:

Definition 9.18 (Soundness (Type-3)). We say that an interpretation is *type-3 sound*, or that an encryption scheme *provides soundness in the type-3 sense*, if the interpretation Φ (resulting from the encryption scheme) is such that

$$\boldsymbol{M \cong_3 N \Rightarrow [\![M]\!]_{\Phi} \approx [\![N]\!]_{\Phi}}$$

for any pair of expressions *M* and *N*.

Definition 9.19 (Completeness (Type-3)). We say that an interpretation is *type-3 complete*, or that an encryption scheme *provides completeness in the type-3 sense*, if the interpretation **Φ** (resulting from the encryption scheme) is such that for any pair of expressions *M* and *N*,

$$\boldsymbol{[\![M]\!]_{\Phi} \approx [\![N]\!]_{\Phi} \Rightarrow M \cong_3 N}.$$

Soundness for Key-Cycles

Below, we present our two main soundness results: if an encryption scheme is KDM secure, it also provides type-3 soundness even in the presence of key-cycles.

Theorem 9.2 (Symmetric KDM Security Implies Soundness). *Let* Π = (*K; E; D*) *be a com-putational symmetric encryption scheme such that* $|\boldsymbol{E}(\boldsymbol{k},\boldsymbol{m},\boldsymbol{w})| = |\boldsymbol{E}(\boldsymbol{k},\boldsymbol{m},\boldsymbol{w}')|$ *for all k* ∈ keys*; m* ∈ plaintexts *and w; w'* ∈ coins. *Then, if the length-function l satisfies only the equalities listed in Definition 2.15, and* Π *is KDM-secure, then* Π *provides type-3 soundness.*

Proof. We first redefine the interpretation of patterns. The only thing we have to change is the interpretation of a box. Now, the interpretation of a pattern $\square_{\boldsymbol{K},\boldsymbol{l}(\mathbf{M})}$ for a given security parameter η is

given by $\boldsymbol{\Phi_{\eta}}(\{\mathbf{0}^{|\Phi_{\eta}(M)|}\}_{K})$. That is, the interpretation function used to encrypt a single 0 under a random key. Now, it encrypts a string of 0s of the same requisite length (length of $\boldsymbol{\Phi_{\eta}}(M)$), and it encrypts them under the correct key $\boldsymbol{\tau}(K)$.

The proof in this case is a somewhat reduced hybrid argument. In a standard hybrid argument, like the one Abadi and Rogaway used to prove their soundness result, several patterns are put between *M* and *N*; then, using security, it is proven that soundness holds between each two consecutive patterns, and therefore soundness holds for *M* and *N*. In our case, we first directly prove that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ is indistinguishable from $[\![\boldsymbol{pattern_3\ M}]\!]_{\boldsymbol{\Phi}}$. Then, since that holds for *N* too, and since $pattern_3(M)$ differs from $pattern_3(N)$ only in the name of keys, $[\![\boldsymbol{pattern_3\ M}]\!]_{\boldsymbol{\Phi}}$is indistinguishable from $[\![\boldsymbol{pattern_3\ N}]\!]_{\boldsymbol{\Phi}}$, therefore the result follows. KDM security is used when we show that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ and $[\![\boldsymbol{pattern_3\ M}]\!]_{\boldsymbol{\Phi}}$ are indistinguishable.

For an arbitrary (formal) key *K*, let *i*(*K*) denote the index of *K*. For an expression *M*, a set of formal (unrecoverable) keys *S*, and a function $\boldsymbol{\tau}: \boldsymbol{Keys \backslash S \rightarrow keys}$, we define a function $\boldsymbol{f_{M,S,\tau} : coins^{e(M)} \times keys^{\infty} \rightarrow strings}$ (where *e*(*M*) is the number of encryptions in *M*) inductively in the following way:

- for $\boldsymbol{M = B \in Blocks, let\ f_{B,S,\tau} :\ keys^{\infty} \rightarrow strings\ be\ defined\ as\ f_{B,S,\tau}(\overline{k}) = B}$;
- for $\boldsymbol{M = K \in Keys \cap S, let\ f_{K,S,\tau} :\ keys^{\infty} \rightarrow strings\ be\ defined\ as\ f_{K,S,\tau}(\overline{k}) = k_{i(K)}}$;
- for $\boldsymbol{M = K \in Keys \cap \overline{S}, let\ f_{K,S,\tau} :\ keys^{\infty} \rightarrow strings\ be\ defined\ as\ f_{K,S,\tau}(\overline{k}) = \tau(K)}$;
- for $\boldsymbol{M = (M_1, M_2), let\ f_{(M_1, M_2),S,\tau} : coins^{e(M_1)} \times coins^{e(M_2)} \times keys^{\infty} \rightarrow strings\ be\ defined\ as\ f_{(M_1,M_2),S,\tau}(\omega M_1, \omega M_2, \overline{k}) = [f_{(M_1),S,\tau}(\omega M_1, \overline{k}), f_{(M_2),S,\tau}(\omega M_2, \overline{k})]}$;

- for $\boldsymbol{M = \{N\}_K \text{ and } K \in S, \text{let } f_{\{N\}_K,S,\tau} : coins \times coins^{e(N)} \times keys^{\infty} \rightarrow strings \text{ be defined as } f_{\{N\}_K,S,\tau}(\omega, \omega_N, \bar{k}) = E\left(k_{i(K)}, f_{N,S,\tau}(\omega_N, \bar{k}), \omega\right)}$.
- for $\boldsymbol{M = \{N\}_K \text{ and } K \notin S, \text{let } f_{\{N\}_K,S,\tau} : coins \times coins^{e(N)} \times keys^{\infty} \rightarrow strings \text{ be defined as } f_{\{N\}_K,S,\tau}(\omega, \omega_N, \bar{k}) = E\left(\tau(K), f_{N,S,\tau}(\omega_N, \bar{k}), \omega\right)}$.

We note that this function is constant length because the keys are constant-length (for the same η) and the length of an encryption only depends on the length of the message and η.

We first prove that $[\![M]\!]_{\Phi} \approx [\![pattern_3(M)]\!]_{\Phi}$. Suppose that $[\![M]\!]_{\Phi} \not\approx [\![pattern_3(M)]\!]_{\Phi}$. This means that there is an adversary A that distinguishes the two distributions, that is

$$Pr\left(x \leftarrow [\![M]\!]_{\Phi_\eta} : A(1^\eta, x) = 1\right) - \mathbf{Pr}(x \leftarrow [\![pattern_3(M)]\!]_{\Phi_\eta} : A(1^\eta, x) = 1)$$

is a non-negligible function of $\boldsymbol{\eta}$. Let us show that this contradicts the fact that the system is (symmetric) KDM-secure. To this end, we can construct an adversary that can distinguish between the oracles $Real_{\bar{k}}$ and $Fake_{\bar{k}}$. From now on, let *S* = Keys \ *R-Keys*(*M*). Consider the following algorithm:

algorithm B^F $(1^\eta, \mathbf{M})$
for K ∈ $R - Keys(M)$ **do** $\tau(K) \leftarrow K(1^\eta)$
y ← CONVERT2(*M*,*M*)
b ← A($1^\eta, y$)
return *b*

algorithm CONVERT2(*M'; M*) with $M' \sqsubseteq M$
if *M'* = *K* where *K* ∈ *R-Keys*(*M*) then
return $\tau(K)$
if *M* = *B* where *B* ∈ Blocks then return *B*
if *M'* = (*M*$_1$*; M*$_2$) then

$x \leftarrow$ CONVERT2($\boldsymbol{M_1}$,M)
$y \leftarrow$ CONVERT2($\boldsymbol{M_2}$,M)
return [x; y]
if $M' = \{\mathbf{M_1}\}_{\mathbf{K}}$ with $K \in$ *R-Keys*(M) then
$x \leftarrow$ CONVERT2($\boldsymbol{M_1}$,M)
$y \leftarrow E(\boldsymbol{\tau}(\boldsymbol{K}), \boldsymbol{x})$
if $M' = \{\mathbf{M_1}\}_{\mathbf{K}}$ with $K \notin$ *R-Keys*(M) then
$\boldsymbol{\omega} \leftarrow \boldsymbol{coins}^{e(\boldsymbol{M_1})}$
$y \leftarrow F(\boldsymbol{i}(\boldsymbol{K}), \boldsymbol{f}_{\boldsymbol{M_1},\boldsymbol{S},\boldsymbol{\tau}}(\boldsymbol{\omega}, .))$
return y

This algorithm applies the distinguisher A ($\mathbf{1}^{\eta}$,•) on the distribution $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ when F is $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$, and the distribution of $[\![\boldsymbol{pattern_3}(\boldsymbol{M})]\!]_{\boldsymbol{\Phi}}$ when F is $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$. So if ($\mathbf{1}^{\eta}$,•) can distringuish $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ and $[\![\boldsymbol{pattern_3}(\boldsymbol{M})]\!]_{\boldsymbol{\Phi}}$, then $\boldsymbol{B}^{\boldsymbol{F}}(\mathbf{1}^{\eta},\bullet)$ can distringuish $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$ and $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$. But we assumed that $\boldsymbol{Real}_{\bar{\boldsymbol{k}}}$ and $\boldsymbol{Fake}_{\bar{\boldsymbol{k}}}$ cannot be distringuished, so $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{pattern_3}(\boldsymbol{M})]\!]_{\boldsymbol{\Phi}}$.

In a similar manner, we can show that $[\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{pattern_3}(\boldsymbol{N})]\!]_{\boldsymbol{\Phi}}$. Finally, it is easy to see that $[\![\boldsymbol{pattern_3}(\boldsymbol{M})]\!]_{\boldsymbol{\Phi}} = [\![\boldsymbol{pattern_3}(\boldsymbol{N})]\!]_{\boldsymbol{\Phi}}$, because the two patterns differ only by key-renaming. Hence $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$.

Let us conclude the consideration of KDM security by demonstrating what Black et al. claimed informally: the notion of KDM security is ‘orthogonal’ to the previous definitions of security. In particular, we can claim that KDM security neither implies nor is implied by type-0 security. The former is proved directly, Theorem 9.4, while the latter is a corollary of previous theorems:

Corollary 9.1. *Type-0 security does not imply (symmetric) KDM-security. If there exists an encryption scheme that is type-0 secure, there exists an encryption scheme which is also type-0 secure but not KDM-secure.*

Proof. Suppose that there exists a type-0 secure encryption scheme. By Theorem 9.1 there is a type-0 secure scheme Π such that Π does not satisfy soundness. If all type-0 encryptions schemes are KDM-secure,

then Π is as well. By Theorem 9.2, this means that Π satisfies soundness—a contradiction.

Theorem 9.4. *KDM security does not imply type-0 security. That is, there is an encryption scheme that is KDM-secure, but not type-0 secure.*

Proof. This is easily seen by inspecting the KDM-secure encryption scheme given by Black et al. in the random oracle model [3]. Let RO be the random oracle, which implements a random function from $\{\mathbf{0},\mathbf{1}\}^*$ to $\{\mathbf{0},\mathbf{1}\}^\infty$. Let Pad ⊕ M and M ⊕ Pad (where M ∈ $\{\mathbf{0},\mathbf{1}\}^*$; 1g□ and Pad ∈ $\{\mathbf{0},\mathbf{1}\}^\infty$) be the bit-wise exclusive-or of M and the first $|\boldsymbol{M}|$ bits of Pad. (Note that $|\boldsymbol{Pad} \oplus \mathbf{M}| = |\boldsymbol{M}|$ exactly.) Let η be the security parameter. Then:

- *K* produces a random bit-string $\boldsymbol{K} \leftarrow \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{\eta}}$;
- The encryption algorithm *E*, on input (*K*, *M*), selects a random bit-string $\boldsymbol{r} \leftarrow \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{\eta}}$ and returns the pair $(\boldsymbol{r}, \boldsymbol{M} \oplus \boldsymbol{RO}(\boldsymbol{r}|\,|\boldsymbol{K}))$;
- D, on input (K; C = $(\boldsymbol{c_1}, \boldsymbol{c_2})$), returns $(\boldsymbol{c_2} \oplus \boldsymbol{RO}(\boldsymbol{c_1}|\,|\boldsymbol{K})$.

This scheme is not type-0 secure because ciphertexts reveal the length of the plaintext. In par-ticular, if *c* is a ciphertext for plaintext *m*, then $|\boldsymbol{c}| = |\boldsymbol{m}| + \boldsymbol{\eta}$. Thus, one can easily distinguish between an oracle that encrypts the input message *m* and an oracle that always encrypts the 1-bit string 0.

9.3 Partial Leakage of Information

In the previous section, we were forced by the definition of KDM security to consider encryption schemes that (possibly) revealed partial information about the plaintext (in particular its length) or the key (such as whether two ciphertexts were made using the same one). For the rest of this discussion, the issue of key-cycles and concentrate our attention upon the issues of such partial leakage was left behind. In particular, let us consider fully general notions of partial leakage. To motive these results, let us present soundness and completeness theorems for two specific examples. In this section, let us separate the leakage of plaintext-length (type-1 encryption) from the leakage of key-sharing

(type-2 encryption) and consider each separately. In particular, let us show in this section that soundness can survive such leakage in the computational model if the formal model is appropriately weakened to match.

Soundness and Completeness for Type-1 Schemes

Let us consider the case of 'type-1' encryption schemes: encryption schemes which may reveal plaintext-length, but which conceals whether or not two ciphertexts were created using the same key. (In the terminology of Abadi and Rogaway, type-1 encryption is message-concealing and which-key concealing, but may be length-revealing.) An equivalent way to express this security definition is that no adversary should be able to tell whether two ciphertexts were created using the same key or different (independent) keys, even if the adversary is allowed to choose the plaintexts, so long as those plaintexts have the same length:

Definition 9.20 (Type-1 Security). Let $\prod$ = (*K; E; D*) be a symmetric encryption scheme. We say that the encryption-scheme is *type-1 secure* if no PPT adversary A can distinguish the pair of oracles ($E(k,\bullet)$, $E(k',\bullet)$)) and $(\boldsymbol{E}(\boldsymbol{k},\boldsymbol{0}^{|\bullet|}),\boldsymbol{E}(\boldsymbol{k},\boldsymbol{0}^{|\bullet|}))$ as k and k' are independently generated, that is, for all PPT adversaries A:

$$\boldsymbol{Pr}\left[\boldsymbol{k},\boldsymbol{k}' \leftarrow \boldsymbol{K}(\boldsymbol{1}^{\boldsymbol{\eta}}): \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\bullet),\boldsymbol{E}(\boldsymbol{k}',\bullet)}(\boldsymbol{1}^{\boldsymbol{\eta}}) = \boldsymbol{1}\right] - \boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\boldsymbol{1}^{\boldsymbol{\eta}}): \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\boldsymbol{0}^{|\bullet|}),\boldsymbol{E}(\boldsymbol{k},\boldsymbol{0}^{|\bullet|})}(\boldsymbol{1}^{\boldsymbol{\eta}}) \leq \boldsymbol{neg}\,(\boldsymbol{\eta})\right]$$

Type-1 security does not provide soundness for the logic of Definition 9.1. For example, one can see immediately that $\{\boldsymbol{0}\}_{\boldsymbol{K_1}} \cong \{\boldsymbol{00}\}_{\boldsymbol{K_1}}$, but $[\![\{\boldsymbol{0}\}_{\boldsymbol{K_1}}]\!]_{\boldsymbol{\Phi}} \not\approx [\![\{\boldsymbol{00}\}_{\boldsymbol{K_1}}]\!]_{\boldsymbol{\Phi}}$ if the encryption scheme reveals the length of the plaintext.

To show soundness or completeness, patterns must reflect those aspects of an expression that an adversary can and cannot see. The idea is similar to the one in Definition 9.16, but now "boxes" are indexed with the only properties leaked by type-1 encryption: the formal length

of the plaintext. (Note, however, that the notions of visible-subexpressions, recoverable keys and formal length remain unchanged.)

Definition 9.21 (Pattern (Type-1)). We define the *set of patterns*, Pat, by the grammar:

$$\text{Pat ::= Keys} \mid \text{Blocks} \mid (\text{Pat; Pat}) \mid \{\boldsymbol{Pat}\}_{\boldsymbol{Keys}} \mid \square_{\boldsymbol{l(\mathbf{Exp})}}$$

The type-1 pattern of an expression M, denoted by $pattern_1(M)$, is derived from M by replacing each term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in \mathbf{vis}\,(\mathbf{M})$ ($\mathbf{where\ K} \notin \mathbf{R - Keys(M)}$) by $\square_{\boldsymbol{l(\mathbf{M'})}}$.

We say that two expressions M and N are *type-1 equivalent*, and denote it by $\boldsymbol{M \cong_1 N}$, if there exists a key-renaming function $\boldsymbol{\sigma}$ $pattern_1(M) =_{\mathbf{1}} pattern_1(N\boldsymbol{\sigma})$ where $=_{\mathbf{1}}$ is defined in the following way:

- if $P \in$ Blocks $\cup$ Keys, then $P =_1 Q$ iff P and Q are identical;
- if P is of the form $\square_{\boldsymbol{l(\mathbf{M'})}}$, then $P =_1 Q$ iff Q is of the form $\square_{\boldsymbol{l(\mathbf{N'})}}$,, and $l(M') = l(N')$ in the sense of Definition 9.15;
- if P is of the form $(P_1; P_2)$, then $P =_1 Q$ iff Q is of the form $(Q_1; Q_2)$ where $P_1 =_1 Q_1$ and $P_2 =_1 Q_2$;
- if P is of the form $\{\boldsymbol{P'}\}_{\boldsymbol{K}}$, then $P =_1 Q$ iff Q is of the form $\{\boldsymbol{Q'}\}_{\boldsymbol{K}}$ where $P' =_1 Q'$.

Again, the symbol $\square_{\boldsymbol{l(\mathbf{M'})}}$ in a pattern reveals that some plaintext is encrypted and its length is $l(M')$.

Example 9.2. Let N be the expression

$$\left(\left(\{\mathbf{0}\}_{\boldsymbol{K_8}}, \{\mathbf{100}\}_{\boldsymbol{K_1}}\right), \left(\boldsymbol{K_7}, \left\{\left(\{\mathbf{101}\}_{\boldsymbol{K_9}}, \{\boldsymbol{K_8}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}, \{\boldsymbol{K_5}\}_{\boldsymbol{K_7}}\right)\right).$$

We have that *R-Keys*(N) = $\{\boldsymbol{K_5}, \boldsymbol{K_7}, \boldsymbol{K_8}\}$, and so, in this case, $pattern_1(N)$ is

$$\left(\left(\{\mathbf{0}\}_{\boldsymbol{K_8}}, \square_{\boldsymbol{l(\mathbf{100})}}\right), \left(\left(\boldsymbol{K_7}, \left\{\left(\square_{\boldsymbol{l(\mathbf{101})}}, \{\boldsymbol{K_8}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}\right), \{\boldsymbol{K_5}\}_{\boldsymbol{K_7}}\right)\right).$$

Defining M as in Example 9.1, $pattern_1(M)$ is

$$\left(\left(\{\mathbf{0}\}_{\boldsymbol{K_6}}, \square_{\boldsymbol{l(\{K_7\}_{K_1})}}\right), \left(\left(\boldsymbol{K_2}, \left\{\left(\square_{\boldsymbol{l(\mathbf{001})}}, \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}\right), \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}\right)\right).$$

Now, if we replace $\boldsymbol{K_6 \to K_8}$, $\boldsymbol{K_2 \to K_7}$ **and** $\boldsymbol{K_5 \to K_5}$ in M, we have that $\boldsymbol{M \cong_1 N}$ **iff** $\boldsymbol{l(\mathbf{100}) = l(\{K_7\}_{K_1})}$.

With these definitions, the following soundness and completeness theorems can be proved.

Theorem 9.5 (Type-1 Soundness). *Let* $\prod$ *be a type-1 secure encryption scheme such that for all k* $\in$ *keys; m* $\in$ *plaintexts and w; w'* $\in$ *coins we have* $|\boldsymbol{E}(\boldsymbol{k},\boldsymbol{m},\boldsymbol{w})| = |\boldsymbol{E}(\boldsymbol{k},\boldsymbol{m},\boldsymbol{w}')|$. *Then, if the length-function satisfies only the equalities defined in Definition 2.15, then for any M and N expressions such that B-Keys(M) and B-Keys(N) are not cyclic in M and N respectively,*

$$\mathbf{M} \cong_{\mathbf{1}} \mathbf{N} \text{ implies } [\![\mathbf{M}]\!]_{\Phi} \approx [\![\mathbf{N}]\!]_{\Phi}.$$

Otherwise, for arbitrary length-function l (that is, one satisfying possible more equations), if for all pairs of expressions M and N, l (M) = l (N) implies that the binary length of $[\![\boldsymbol{M}]\!]_{\Phi_\eta}$ *is the same as the binary length of* $[\![\boldsymbol{N}]\!]_{\Phi_\eta}$ *for each security parameter ′, then for any M and N expressions,*

$$\boldsymbol{M} \cong_{\mathbf{1}} \boldsymbol{N} \text{ implies } [\![\boldsymbol{M}]\!]_{\Phi} \approx [\![\boldsymbol{N}]\!]_{\Phi}.$$

In addition to soundness, also let us demonstrate the completeness. If soundness shows that formal in-distinguishability implies computational indistinguishability, completeness shows the converse. Rephrased, completeness implies that formal *distinguishability* (as opposed to *in*distinguishability) implies computational distinguishability. For this to be true, the interpretation function must en-force a handful of 'atomic' distinguishability properties:

Theorem 9.6 (Type-1 Completeness). *Let* $\prod$ *be a type-1 secure encryption scheme such that* $|\boldsymbol{E}(\boldsymbol{k},\boldsymbol{m},\boldsymbol{w})| = |\boldsymbol{E}(\boldsymbol{k},\boldsymbol{m},\boldsymbol{w}')|$ *for all k* $\in$ keys*; m* $\in$ plaintexts *and w; w'* $\in$ coins*. We have:*

$$[\![\boldsymbol{M}]\!]_{\Phi} \approx [\![\boldsymbol{N}]\!]_{\Phi} \text{ implies } \boldsymbol{M} \cong_{\mathbf{1}} \boldsymbol{N}$$

for all M and N pairs of expressions if and only if the following conditions hold: for any

K; K'; K'' ∈ Keys, B ∈ Blocks, M; M'; N ∈ Exp,

(i) *no pair of* $[\![\boldsymbol{K}]\!]_{\Phi}, [\![\boldsymbol{B}]\!]_{\Phi}, [\![\boldsymbol{K}(\boldsymbol{M},\boldsymbol{N})]\!]_{\Phi}, [\![\{\boldsymbol{M}'\}_{\boldsymbol{K}'}]\!]_{\Phi}$ *are equivalent with respect to* $\approx$*,*

(ii) *if* $[\![(\boldsymbol{K},\{\boldsymbol{M}\}_{\boldsymbol{K}})]\!]_{\Phi} \approx [\![(\boldsymbol{K}'',\{\boldsymbol{M}'\}_{\boldsymbol{K}'},)]\!]_{\Phi}$ *, then* $K' = K''$*, and*

(iii) *if* $[\![\{\boldsymbol{M}\}_{\boldsymbol{K}}]\!]_{\Phi} \approx [\![\{\boldsymbol{M}'\}_{\boldsymbol{K}'}]\!]_{\Phi}$ *then l*(*M*) = *l* (*M'*).

Some aspects of this theorem merit further discussion. First, note that the theorem does not mention key-cycles. Secondly, note that Condition (i) requires that different types of ob-jects, blocks, keys, pairs and encryption terms should be distinguishable to achieve completeness; this can be ensured by tagging each object with its type, as suggested in [2]. Thirdly, Condition (ii) (which we call *weak confusion-freeness*) is equivalent to the property of weak key-authenticity introduced by Horvitz and Gligor [6] in the case of type-0 schemes. This property essentially means that decrypting with the wrong key should be detectable in a proba-bilistic sense. Finally, condition (iii) requires that encryption of messages with different length should be detectable. Definition 2.20 *allows* that encryptions of messages of different length may be detected but does not enforce it. That suffices for soundness, but completeness requires that it should be detectable when ciphertexts contain messages of different lengths. A purely computational condition that implies condition (iii) is the notion of *strictly length revealing*:

Definition 9.22 (Strictly Length Revealing Scheme). Let ∏ = (*K; E; D*) be a symmetric en-cryption scheme. We say that the encryption-scheme is *strictly length revealing* if it is type-1 secure but there exists a PPT adversary A such that the following function is a non-negligible function of η:

$$\boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\bullet)}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] - \boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\mathbf{0}^{\neq|\bullet|})}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right]$$

We use $\mathbf{0}^{\neq|\bullet|}$ to denote 0^n, where $\boldsymbol{n} \neq |\bullet|$.

Soundness and Completeness for Type-2 Schemes

Having considered the leakage of plaintext-length in the previous section, let us turn to the other to the kinds of leakage seen in KDM-security: whether or not two ciphertext share a key. However, we can now assume that the plaintext conceals the plaintext-length. ('Type-2' in the terminology of Abadi and Rogaway, as well as message-concealing, length-concealing, and which-key reveal-ing.) For this type

of encryption, no adversary should be able to tell whether a ciphertext contains a (possibly long) plaintext or the single-bit plaintext 0:

Definition 9.23 (Type-2 Security). Let $\prod$ = (*K; E; D*) be a symmetric encryption scheme. We say that the encryption-scheme is *type-2 secure* if no PPT adversary A can distinguish the oracles $E(k; \bullet)$ and $E(k; 0)$ as *k* is randomly generated, that is, for all PPT adversaries A:

$$\boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\boldsymbol{1^{\eta}}) : \boldsymbol{A^{E(k,\bullet)}}(\boldsymbol{1^{\eta}}) = \boldsymbol{1}\right] - \boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\boldsymbol{1^{\eta}}) : \boldsymbol{A^{E(k,0)}}(\boldsymbol{1^{\eta}}) = \boldsymbol{1}\right] \leq \boldsymbol{neg}(\boldsymbol{\eta})$$

Again, patterns must be re-defined to reflect all the information about an expression which may be available to the adversary, but only that information:

Definition 9.24 (Pattern (Type-2)). We define the *set of patterns*, Pat, by the grammar:

Pat ::= Keys | Blocks | (Pat; Pat) | $\{\boldsymbol{Pat}\}_{\boldsymbol{Keys}}$ | $\square_{\boldsymbol{Keys}}$

The type-2 pattern of expression *M*, denoted be $pattern_2(M)$, is derived from *M* by replacing each term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in vis\ (M)$ (where K ∉ R-Keys(M)) by $\square_{\boldsymbol{K}}$.

We say that two expression M and N are type-2 equivalent, and denote it by $\boldsymbol{M} \cong_{\boldsymbol{2}} \boldsymbol{N}$, if there exist a key-renaming function σ such that $\boldsymbol{pattern_2(M)} =_{\boldsymbol{2}} \boldsymbol{pattern_2(N\sigma)}$ where $=_{\boldsymbol{2}}$ is defined in the following way:[2]

- if $P \in$ Blocks ∪ Keys, then $P =_2 Q$ iff *P* and *Q* are identical;
- if *P* is of the form $\square_{\boldsymbol{K}}$, then $P =_2 Q$ iff *Q* is also of the form $\square_{\boldsymbol{K}}$;
- if *P* is of the form $(P_1; P_2)$, then $P =_2 Q$ iff *Q* is of the form $(Q_1; Q_2)$ where $P_1 =_2 Q_1$ and $P_2 =_2 Q_2$.;
- if *P* is of the form $\{\boldsymbol{P'}\}_{\boldsymbol{K}}$, then $P =_2 Q$ iff *Q* is of the form $\{\boldsymbol{Q'}\}_{\boldsymbol{K}}$, where $P' =_2 Q'$.

Example 9.3. Let *N* be the same expression as in Example 9.2,

$$\left(\left(\{\boldsymbol{0}\}_{\boldsymbol{K_8}}, \{\boldsymbol{100}\}_{\boldsymbol{K_1}}\right), \left(\left(\boldsymbol{K_7}\left\{\left(\{\boldsymbol{101}\}_{\boldsymbol{K_9}}, \{\boldsymbol{K_8}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}\right), \{\boldsymbol{K_5}\}_{\boldsymbol{K_7}}\right)\right).$$

We have that *R-Keys*(*N*) = $\{\boldsymbol{K_5}, \boldsymbol{K_7}, \boldsymbol{K_8}\}$, and so, in this case, *pattern*$_2$(*N*) is

$$\left(\left(\{\mathbf{0}\}_{\boldsymbol{K_8}}, \square_{\boldsymbol{K_1}}\right), \left(\left(\boldsymbol{K_7}\left\{\left(\square_{\boldsymbol{K_9}}, \{\boldsymbol{K_8}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}\right), \{\boldsymbol{K_5}\}_{\boldsymbol{K_7}}\right)\right).$$

Defining *M* as in Example 9.1, *pattern*$_2$(*M*) is

$$\left(\left(\{\mathbf{0}\}_{\boldsymbol{K_6}}, \square_{\boldsymbol{K_4}}\right), \left(\left(\boldsymbol{K_2}\left\{\left(\square_{\boldsymbol{K_3}}, \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}\right), \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}\right)\right).$$

Now, if we replace $\boldsymbol{K_6} \rightarrow \boldsymbol{K_8}, \boldsymbol{K_4} \rightarrow \boldsymbol{K_1}, \boldsymbol{K_2} \rightarrow \boldsymbol{K_7}, \boldsymbol{K_3} \rightarrow \boldsymbol{K_9}$, and $\boldsymbol{K_5} \rightarrow \boldsymbol{K_5}$, in M , we have that $\boldsymbol{M} \cong_2 \boldsymbol{N}$.

With these definitions, the following soundness and completeness theorems can be proved.

Theorem 9.8 (Type-2 Completeness). *Let* $\prod$ *be a type-2 secure encryption scheme. We have*

$$[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}} \textbf{ implies } \boldsymbol{M} \cong_2 \boldsymbol{N}$$

that, for any pairs of expressions M and N if and only if the following conditions hold: for any

K; K$'$*; K*$''$ ∈ *Keys, B* ∈ *Blocks, M; M*$'$*; N; N*$'$ ∈ *Exp,*

(i) *no pair of* $[\![\boldsymbol{K}]\!]_{\boldsymbol{\Phi}}, [\![\boldsymbol{B}]\!]_{\boldsymbol{\Phi}}, [\![(\boldsymbol{M}, \boldsymbol{N})]\!]_{\boldsymbol{\Phi}}, [\![\{\boldsymbol{M'}\}_{\boldsymbol{K'}}]\!]_{\boldsymbol{\Phi}}$ *are equivalent with respect to*≈*;*

(ii) *if* $[\![(\boldsymbol{K}, \{\boldsymbol{M}\}_{\boldsymbol{K}})]\!]_{\boldsymbol{\Phi}} \approx [\![(\boldsymbol{K''}, \{\boldsymbol{M'}\}_{\boldsymbol{K'}})]\!]_{\boldsymbol{\Phi}}$*, then K' = K'';*

(iii) *if* $[\![(\{\boldsymbol{M}\}_{\boldsymbol{K}}, \{\boldsymbol{M'}\}_{\boldsymbol{K}}]\!]_{\boldsymbol{\Phi}} \approx [\![(\{\boldsymbol{N}\}_{\boldsymbol{K'}}, \{\boldsymbol{N'}\}_{\boldsymbol{K''}}]\!]_{\boldsymbol{\Phi}}$ *then K' = K''.*

The conditions of the completeness theorem are similar to the ones for the type-1 case ex-cept for condition (iii). This condition requires that encryption with different keys should be detectable. Definition 9.23 *allows* that encrypting with different keys may be detectable, but it does not *require* it. That suffices for soundness, but such detection *is* required for completeness. It is easily shown that condition (iii) is implied by the purely computational definition of a *strictly key revealing* encryption scheme:

Definition 9.25 (Strictly Key Revealing Scheme). Let $\prod$ = (*K; E; D*) be a symmetric encryp-tion scheme. We say that the encryption-scheme is *strictly key revealing* if it is type-2 secure but there exists a

PPT adversary A such that the following function is a non-negligible function of η:

$$\boldsymbol{Pr}\left[\boldsymbol{k}, \boldsymbol{k'} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\bullet),\boldsymbol{E}(\boldsymbol{k'},\bullet)}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] - \boldsymbol{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\bullet),\boldsymbol{E}(\boldsymbol{k},\bullet)}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right]$$

9.3.3 Soundness and Completeness for Type-3 Schemes

Type-3 encryption schemes (Definition 9.14, also called message-concealing, which-key reveal-ing and length-revealing in the terminology of Abadi and Rogaway) can be thought of as leaking the information leaked by both type-1 and type-2 schemes. Both soundness and completeness results follow using the notion of patterns from Definition 9.16. As with type-1 and type-2 en-cryption, completeness requires that it is possible to distinguish ciphertexts that were encrypted with different keys, and to distinguish ciphertexts for which the plaintexts have different lengths. That is, the encryption scheme must be both strictly key revealing *and* strictly length revealing (Definitions 2.25 and 2.22 respectively).

9.4 Information-Theoretic Interpretations: Soundness and Completeness for One-Time Pad

Besides the computational definition, there are other possible important notions of 'indistin-guishability.' For example, we could say that two distributions are 'indistinguishable' if and only if they are *identical*. Such a notion would lead to new (but analogous) notions of soundness and completeness, and we can explore these new notions using (as a specific encryption scheme) the One-Time Pad (OTP).

Let strings $\coloneqq \{\mathbf{0}, \mathbf{1}\}^*$with the following pairing function: For any two strings *x; y* ∈ strings we can define the pairing of *x* and *y* as $[\boldsymbol{x}, \boldsymbol{y}] \coloneqq \langle \boldsymbol{x}, \boldsymbol{y}, \mathbf{0}, \mathbf{1}_{|\boldsymbol{y}|} \rangle$ $\boldsymbol{where}$ $\langle , , \ldots , \rangle i$ denotes the concatenation of the strings separated by the commas, 1_m stands for *m* many 1's, and for any *x* $\boldsymbol{x} \in \{\mathbf{0}, \mathbf{1}\}^*, |\boldsymbol{x}|$denotes the length of the string. The number of 1's at the end indicate how long the second string is in the pair, and the 0 separates the strings from the 1's. Let blocks be those strings that end

with 001. The ending is just a tag, it shows that the meaning of the string is a block.

Key-Generation. In case of the OTP, the length of the encrypting key must match the length of the plaintext. Thus, we need a separate key-generation for each length. That is, for each $n > 3$, K_n is a random variable over some discrete probability field $(\Omega_{\mathcal{K},n}, \Pr_{\mathcal{K},n})$ such that its values are equally distributed over $\boldsymbol{keys_n} \coloneqq \{\boldsymbol{k} \mid \boldsymbol{k} \in \boldsymbol{strings}, |\boldsymbol{k}| = \boldsymbol{n}, \boldsymbol{k\ ends\ with}\ \mathbf{010}\}$. Let $\boldsymbol{keys} \coloneqq \cup_{\boldsymbol{4}}^{\infty}\ \boldsymbol{keys_n}$. For $k \in$ keys, let core(k) denote the string that we get from k by cutting the tag 010.

Encryption. Let the domain of the encryption function, Dom_E , be those elements $(k; x) \in$ keys $\times$ strings, for which $|\boldsymbol{k}| = |\boldsymbol{x}| + \mathbf{3}$, and let $\boldsymbol{E(k,x)} \coloneqq \langle \boldsymbol{core(k)} \oplus \mathbf{x}, \mathbf{110} \rangle$. The tag 110 informs us that the string is a ciphertext. Notice that this encryption is not probabilistic, hence $\boldsymbol{E(k,x)}$ is not a random variable. Notice also, that the tag of the plaintext is not dropped, that part is also encrypted.

Decryption. The decryption function $\boldsymbol{D(k,x)}$ is defined whenever $|\boldsymbol{k}| = |\boldsymbol{x}|$, and, naturally the value of $\boldsymbol{D(k,x)}$ is the first $|\boldsymbol{k}| - \mathbf{3}\ \boldsymbol{bits\ of\ k} \oplus \mathbf{x}$.

Indistinguishability. Let us now call two distributions indistinguishable, if they are identical, and denote this relation by $=_d$.

As in the case of type-3 encryption, lengths of the messages are revealed. Therefore, we must again define the *length* of an expression.

Definition 9.26. We assume that some length function $\boldsymbol{l} : \boldsymbol{Keys} \to \{\mathbf{4}, \mathbf{5}, \ldots\}$ is given on the keys symbols. The length of a block is defined as $\boldsymbol{l(B)} \coloneqq |\boldsymbol{B}| + \mathbf{3}$. We added 3 to match the length of the tag. We define the length function on any expression in Exp by induction:

- $\boldsymbol{l}\big((\boldsymbol{M},\boldsymbol{N})\big) \coloneqq \boldsymbol{l(M)} + \mathbf{2}\boldsymbol{l(N)} + \mathbf{1}$,
- $\boldsymbol{l}(\{\boldsymbol{M}\}_{\boldsymbol{K}}) \coloneqq \boldsymbol{l(M)} + \mathbf{3}, \boldsymbol{if}\ \boldsymbol{l(M)} = \boldsymbol{l(K)} - \mathbf{3}, \boldsymbol{and}$
- $\boldsymbol{l}(\{\boldsymbol{M}\}_{\boldsymbol{K}}) \coloneqq \mathbf{0}, \boldsymbol{if}\ \boldsymbol{l(M)} \neq \boldsymbol{l(K)} - \mathbf{3}$.

The *valid expressions* are defined as those expressions in which the length of the encrypted subexpressions match the length of the encrypting key, and, in which no key is used twice to en-crypt. (This

latter condition is necessary to prevent leaking information because of the properties of the OTP.)

Definition 9.27. We define the *valid expressions for OTP* as

$$\boldsymbol{Exp_{OTP}} = \left\{ \begin{array}{l} \boldsymbol{M} \in \boldsymbol{Exp} \mid \boldsymbol{M'} \sqsubseteq \boldsymbol{M} \textbf{ implies } \boldsymbol{l}(\boldsymbol{M'}) > \boldsymbol{0}, \\ \textbf{and each key encrypts at most once in } \boldsymbol{M} \end{array} \right\}.$$

The interpretation function for the OTP is defined similarly to the other cases, with some mi-nor changes regarding the tagging of the messages. Also, there is no security parameter in this encryption scheme, so the interpretation outputs a single random variable for each formal ex-pression (rather than a family of such variables). Let us consider the full algorithm:

algorithm $\boldsymbol{INTERPRETATION_{OTP}}(\boldsymbol{M})$
for $K \in Keys(M)$ do $\boldsymbol{\tau}(\boldsymbol{K}) \leftarrow \boldsymbol{K_{l(K)}}$ $\boldsymbol{y} \leftarrow \boldsymbol{CONVERT_{OTP}}(\boldsymbol{M})$
return y
algorithm $\boldsymbol{CONVERT_{OTP}}(\boldsymbol{N})$
if $N = K$ where $K \in$ Keys then return $\boldsymbol{\tau}(K)$
if $N = B$ where $B \in$ Blocks then return $\langle \boldsymbol{B};\ \boldsymbol{100} \rangle$
if $N = (N_1; N_2)$ then
return $[\boldsymbol{CONVERT_{OTP}}(\boldsymbol{N_1}); \boldsymbol{CONVERT_{OTP}}(\boldsymbol{N_2})]$
if $N = \{\boldsymbol{N_1}\}_{\boldsymbol{K}}$ then
return $\langle \boldsymbol{E}\big(\boldsymbol{\tau}(\boldsymbol{K}), \boldsymbol{CONVERT_{OTP}}(\boldsymbol{N_1})\big), \boldsymbol{110} \rangle$

As in the previous cases, we must again find a suitable equivalence relation for formal expres-sions. One possibility is to index the boxes again with the encrypting keys. Another possibility is to label the boxes with the length as well, but in the OTP scheme, the key reveals the length of the ciphertext. Therefore, we can use the first, that is a simpler possibility. Thus OTP-patterns are defined as follows:

Definition 9.28 (Pattern (OTP)). We define the *set of patterns*, Pat, by the grammar:

Pat ::= Keys | Blocks | (Pat; Pat) | $\{\boldsymbol{Pat}\}_{\boldsymbol{Keys}}$ | $\square_{\boldsymbol{Keys}}$

The OTP pattern of a valid expression *M*, denoted by $\boldsymbol{pattern_{OTP}(M)}$, is derived from *M* by replacing each term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \in \boldsymbol{vis}\ (\boldsymbol{M})$ (where $K \notin$ *R-Keys*(*M*)) by $\square_{\boldsymbol{K}}$.

We say that two expressions and are *OTP equivalent*, and denote it by $\cong_{\boldsymbol{OTP}}$*N*, if there exists a length-preserving key-renaming function $\boldsymbol{\sigma}$ such that $pattern_{\text{OTP}}(M) =_2 pattern_{\text{OTP}}(N\,\boldsymbol{\sigma})$ with $=_2$ as in Definition 9.24

Then, then following soundness and completeness theorems can be proved.

Theorem 9.9 (OTP Soundness). *Let M and N be two valid expressions in* Exp_{OTP} *such that B-Keys*(*M*) *and B-Keys*(*N*) *are not cyclic in M and N respectively. Then,* $\boldsymbol{M \cong_{OTP} N}$ *implies that* $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ ***and*** $[\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$ *are the same probability distributions.*

Theorem 9.10 (OTP Completeness). *Let M and N be two valid expressions in* $\boldsymbol{Exp_{OTP}}$. *Then if* $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ ***and*** $[\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$ *have the same probability distributions, we have that* $\boldsymbol{M \cong_{OTP} N}$.

Note that the completeness theorem for OTP does not contain any side conditions like those of Theorems 9.6 and 9.8. This is because here, what would have been condition (i) from Theo-rem 9.6 is immediate due to the tagging. The natural condition (ii) also follows from the tagging since decrypting with the wrong key will result in a meaningless text. The natural Condition (iii) is meaningless in this case since we just encrypt at most once with each key.

9.5 A General Treatment for Symmetric Encryption

In this section, let us provide a general treatment of soundness and completeness for the Abadi-Rogaway type logics of formal encryptions. The following contain the cases discussed in the pre-vious two sections as special cases. Let us consider a general probabilistic framework for symmetric encryptions, which includes both the computational and the information-theoretic encryption schemes. Then let us show a general way to handle partial leaking of encryption in the formal view. This will be done essentially via an equivalence relation on the set of encryption terms, which is meant to express which encryption terms are

in-distinguishable for an adversary. Also let us introduce the important notion that we call *properness* of this equivalence relation. This is essential, because this is exactly the property that will make an Abadi-Rogaway type hybrid argument go through. Finally, this section presents the interpretation, the general soundness and completeness results and how the theorems for the type-1, type-2 and OTP cases that were presented before follow from the general theorems.

A General Treatment for Symmetric Encryptions

Let us provide a general probabilistic framework for symmetric encryption, which contains both the computational and the information-theoretic description as special cases. Keys, plaintexts and ciphertexts are elements of some discrete set $\overline{\boldsymbol{strings}}$. This is $(\{\mathbf{0},\mathbf{1}\}^*)^\infty$ in the case of a computational treatment, and it is $\{\mathbf{0},\mathbf{1}\}^*$ for the information-theoretic description. The ele-ments of $(\{\mathbf{0},\mathbf{1}\}^*)^\infty$ are sequences in $\{\mathbf{0},\mathbf{1}\}^*$, corresponding to a parameterisation by the security parameter.

A fixed subset, $\overline{\boldsymbol{plaintext}} \subseteq \overline{\boldsymbol{strings}}$ represents the messages that are allowed to be en-crypted. Another subset, $\overline{\boldsymbol{keys}} \subseteq \overline{\boldsymbol{strings}}$ is the possible set encrypting keys that corresponds to the range of the key generation algorithm *K*. In order to be able to build up longer messages from shorter ones, let us assume that an injective *pairing function is* given: $[.,.] : \overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}} \to \overline{\boldsymbol{strings}}$. The range of the pairing function will be called $\overline{\boldsymbol{pairs}}$: $\overline{\boldsymbol{pairs}} := \boldsymbol{Ran}_{[.,.]}$. A symmetric encryption scheme has the following constituents:

Key-generation. Key-generation is represented by a random variable $\boldsymbol{K} : \boldsymbol{\Omega_K} \to \overline{\boldsymbol{keys}}$, over a discrete probability field $(\boldsymbol{\Omega_K}, \boldsymbol{Pr_K})$. In a given scheme, more than one key-generation is allowed.

Encryption. For a given $k \in \overline{\boldsymbol{keys}}$, and a given $x \in \overline{\boldsymbol{plaintext}}$, $E(k; x)$ is a random variable over some discrete probability field $(\boldsymbol{\Omega_E}; \boldsymbol{Pr_E})$. The values of this random variable are in strings and are denoted by $E(\boldsymbol{k},\boldsymbol{x})(\boldsymbol{\omega})$, whenever $\in \boldsymbol{\Omega_E}$.

Decryption. An encryption must be decryptable, so we assume that for each $\boldsymbol{k} \in \boldsymbol{keys}$, a function $\boldsymbol{D} : (\boldsymbol{k}, \boldsymbol{x}) \mapsto \boldsymbol{D}(\boldsymbol{k}, \boldsymbol{x})$ is given satisfying $\boldsymbol{D_k}\big(\boldsymbol{E}(\boldsymbol{k}, \boldsymbol{x})(\boldsymbol{\omega})\big) = \boldsymbol{x}$ **for all** $\boldsymbol{\omega} \in \boldsymbol{\Omega_E}$ **and** $\boldsymbol{x} \in \overline{\boldsymbol{plaintext}}$.

The notion of *indistinguishability* is important both in case of computational and information-theoretic treatments of cryptography. It expresses when there is only very small probability to tell two probability distributions apart.

Indistinguishability. We assume that an equivalence relation called *indistinguishability* is defined on distributions over $\overline{\boldsymbol{strings}}$. We will denote this relation by ≈. We will also say that two random variables taking values in $\overline{\boldsymbol{strings}}$ are equivalent (indistinguishable) if (and only if) their distributions are equivalent; we will use ≈ for denoting this equivalence between random variables as well. For ≈, we require the followings:

(i) Random variables with the same distribution are indistinguishable;

(ii) Constant random variables are indistinguishable if and only if the constants are the same;

(iii) For random variables $\boldsymbol{F} : \boldsymbol{\Omega_F} \to \overline{\boldsymbol{strings}}$ and $\boldsymbol{G} : \boldsymbol{\Omega_G} \to \overline{\boldsymbol{strings}}$, if $F \approx G$, the following must hold: If $\boldsymbol{\pi^i}$ denotes the projection onto one of the components of $\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}$, then $\boldsymbol{\pi^i}\ \mathbf{O}\ [\bullet\,,\ \bullet]^{-1}\mathbf{O}\ \mathbf{F} \approx \boldsymbol{\pi^i}\ \mathbf{O}\ [\bullet\,,\ \bullet]^{-1}\ \mathbf{O}\ \ \boldsymbol{G}\ \boldsymbol{for}\ \boldsymbol{i} = \mathbf{1}, \mathbf{2}$;

(iv) If $\boldsymbol{F'} : \boldsymbol{\Omega_F} \to \overline{\boldsymbol{string}}$, $\boldsymbol{G'} : \boldsymbol{\Omega_G} \to \overline{\boldsymbol{string}}$ are also indistinguishable random variables such that F and $\boldsymbol{F'}$ are independent and G and $\boldsymbol{G'}$ are also independent, then $\boldsymbol{\omega_F} \mapsto [\mathbf{F}(\boldsymbol{\omega_F}),\ \mathbf{F'}(\boldsymbol{\omega_F})]$ and $\boldsymbol{\omega_G} \mapsto [\mathbf{G}(\boldsymbol{\omega_G}),\ \mathbf{G'}(\boldsymbol{\omega_G})]$ are indistinguishable random variables; more-over, if $\boldsymbol{\alpha}, \boldsymbol{\beta} : \overline{\boldsymbol{strings}} \to \overline{\boldsymbol{strings}}$ are functions that preserve ≈ ($\boldsymbol{i.e.}\ \boldsymbol{\alpha}\ \mathbf{o}\ \mathbf{F} \approx \boldsymbol{\alpha}\ \mathbf{o}\ \mathbf{G}$ **and** $\boldsymbol{\beta}\ \mathbf{o}\ \mathbf{F}\ \boldsymbol{\beta}\ \mathbf{o}\ \mathbf{G}$ **whenever** $\mathbf{F} \approx \mathbf{G}$) then $\boldsymbol{\omega_F} \mapsto [(\boldsymbol{\alpha}\ \mathbf{o}\ \mathbf{F})(\boldsymbol{\omega_F}), (\boldsymbol{\beta}\ \mathbf{o}\ \mathbf{F})(\boldsymbol{\omega_F})]$ and $\boldsymbol{\omega_G} \mapsto [(\boldsymbol{\alpha}\ \mathbf{o}\ \mathbf{G})(\boldsymbol{\omega_G}), (\boldsymbol{\beta}\ \mathbf{o}\ \mathbf{G})(\boldsymbol{\omega_G})]$ are indistinguishable random variables if $\boldsymbol{F} \approx \boldsymbol{G}$.

Indistinguishability needs to satisfy some further properties under encryption and decryption that we will specify under the definition of encryption schemes below.

Example 9.4. The simplest example for indistinguishability is that it holds between two random variables if and only if their distributions are identical.

Example 9.5. The standard notion of computational indistinguishability in [7] is also a special case of the general definition. In this case $\overline{\boldsymbol{strings}} = (\{\mathbf{0},\mathbf{1}\}^*)^\infty = \boldsymbol{strings}^\infty$. Ran-dom variables of computational interest have the form $\boldsymbol{F} : \boldsymbol{\Omega_F} \to \boldsymbol{strings}^\infty$ and have inde-pendent components; *i.e.*, for $\eta \in \mathbb{N}$ security parameter, denoting the η'th component of *F* by $\boldsymbol{F_\eta} : \boldsymbol{\Omega_F} \to \boldsymbol{strings}^\infty$, it is required that $\boldsymbol{F_\eta}$ and $\boldsymbol{F_{\eta'}}$ are independent random variables for $\boldsymbol{\eta \neq \eta'}$. Indistinguishability then is phrased with the ensemble of probability distributions of the components of the random variables.

Definition 9.29. An *encryption scheme* is a quadruple $\Pi = (\{\boldsymbol{K_i}\}_{\boldsymbol{i\in I}}, \boldsymbol{E}, \boldsymbol{D}, \approx)$ where $\{\boldsymbol{K_i}\}_{\boldsymbol{i\in I}}$ is a set of key-generations for some index set *I*, *E* is an encryption, *D* decrypts ciphertexts encrypted by *E*, and ≈ is the indistinguishability defined above. We require that for any $\boldsymbol{i} \in I$, the probability distribution of K_i be distinguishable from any constant in $\overline{\boldsymbol{strings}}$, the distributions of K_i and of K_j be distinguishable whenever $\boldsymbol{i \neq j}$, and also that the distribution of (*k; k'*) be distinguishable from the distribution of (*k; k'*) if *k* and *k'* are independently generated: $\boldsymbol{k \leftarrow K_i, k' \leftarrow K_j}$ for any $\boldsymbol{i,j \in I}$. The indistinguishability relation ≈, besides satisfying the properties stated before, needs to be such that if *F* and *G* are random variables taking values in $\overline{\boldsymbol{strings}}$, and K_i is a key-generation such that the distribution of [K_i*; F*] is indistinguishable from the distribution of [K_i*; G*], then:

- $(\boldsymbol{\omega_E}, \boldsymbol{\omega_{K,i}}, \boldsymbol{\omega}) \mapsto \mathbf{E}(\mathbf{K_i}(\boldsymbol{\omega_{K,i}}), \mathbf{F}(\boldsymbol{\omega}))(\boldsymbol{\omega_E})$ and $(\boldsymbol{\omega_E}, \boldsymbol{\omega_{K,i}}, \boldsymbol{\omega}) \mapsto \mathbf{E}(\mathbf{K_i}(\boldsymbol{\omega_{K,i}}), \mathbf{G}(\boldsymbol{\omega}))(\boldsymbol{\omega_E})$ are indistinguishable random variables;
- $(\boldsymbol{\omega_{K,i}}, \boldsymbol{\omega}) \mapsto \mathbf{D}(\mathbf{K_i}(\boldsymbol{\omega_{K,i}}), \mathbf{F}(\boldsymbol{\omega}))$ and $(\boldsymbol{\omega_{K,i}}, \boldsymbol{\omega})$ are also indistinguishable random variables.

Here the probability over $\boldsymbol{\Omega_{K_i}} \times \boldsymbol{\Omega_F}$ is the joint probability of K_i and F , which are here not necessarily independent. Similarly for G.

Equivalence of Expressions

In their treatment, Abadi and Rogaway defined equivalence of expressions via replacing encryp-tion terms encrypted with non-recoverable keys in an expression by a box; two expressions then were declared equivalent if once these encryption terms were replaced, the obtained *patterns* looked the same up to *key-renaming*. This method implicitly assumes, that an adversary cannot distinguish any undecryptable terms. However, if we want to allow leakage of partial information, we need to modify the notion of equivalence.

Before introducing our notion of equivalence of expressions, let us postulate an equivalence notion $\equiv_{\boldsymbol{K}}$ on the set of keys, and another equivalence, $\equiv_{\boldsymbol{C}}$ on the set of *valid* encryption terms. The word *valid*, defined precisely below, is meant for those encryption terms (and expressions) that "make sense". Then, the equivalence on the set of valid expressions will be defined with the help of $\equiv_{\boldsymbol{K}}$ and $\equiv_{\boldsymbol{C}}$.

The reason for postulating equivalence on the set of keys is that there is need to allow many key-generation processes in the probabilistic setting. We therefore have to be able to distinguish formal keys that were generated by different key-generation processes. Therefore, we assume that an equivalence relation $\equiv_{\boldsymbol{K}}$ is given on the set of keys such that each equivalence class contains infinitely many keys. Let $\boldsymbol{Q_{Keys}} \coloneqq \boldsymbol{Keys}/\equiv_{\boldsymbol{K}}$.

Definition 9.30 (Key-Renaming Function). A bijection $\boldsymbol{\sigma} : \boldsymbol{Keys} \to \boldsymbol{Keys}$ is called *key-renaming function*, if $\boldsymbol{\sigma}(\boldsymbol{K}) \equiv_{\boldsymbol{K}}$ K for all $K \in$ Keys. For any expression M, $M\,\boldsymbol{\sigma}$ denotes the expression obtained from M by replacing all occurrences of keys K in M by $\boldsymbol{\sigma}$ (K).

Definition 9.31. We define the *support* of a key-renaming function σ, and denote it by $supp(\boldsymbol{\sigma})$, as the subset of Keys such that $\boldsymbol{\sigma}$ (K) $\neq K$.

We say that two key-renaming functions $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ are *compatible* if for all keys $K \in supp(\boldsymbol{\sigma}) \cap supp(\boldsymbol{\tau})$ we have that $\boldsymbol{\sigma}$ (K) = $\boldsymbol{\tau}$ (K).

The set Exp is often too big to suit our purposes. For example, sometimes we require that certain messages can be encrypted with certain keys only. We therefore define the set of valid expressions:

Definition 9.32. A set of *valid expressions* is a subset $\boldsymbol{Exp_v}$ of Exp such that:

(i) all keys and all blocks are contained in $\boldsymbol{Exp_v}$;

(ii) if $M \in \boldsymbol{Exp_v}$ then *sub*(M) $\subset \boldsymbol{Exp_v}$ and any number of pairs of elements in *sub*(M) are also in $\boldsymbol{Exp_v}$;

(iii) for any key-renaming function $\boldsymbol{\sigma}$, $M \in \boldsymbol{Exp_v}$ iff $M\boldsymbol{\sigma} \in \boldsymbol{Exp_v}$.

Given a set of valid expressions, the set of *valid encryption terms* is $\boldsymbol{Enc_v} :=$ Enc $\cap$ $\boldsymbol{Exp_v}$.

Given a set of valid expressions, the set of *valid encryption terms* is $\boldsymbol{Enc_v} :=$ Enc $\cap$ $\boldsymbol{Exp_v}$.

Equivalence of valid expressions is meant to incorporate the notion of security into the model: two expressions have to be equivalent when they look the same to an adversary. If the encryption is so secure that no partial information is revealed, then all undecryptable terms should look the same to an adversary. If partial information, say repetition of the encrypting key, or length is revealed, then the notion of equivalence accordingly have to be adjusted. This can be done by introducing an equivalence relation on the set of valid encryption terms in order to capture which ciphertexts an adversary can and cannot distinguish; in other words, what partial information (length, key, etc...) can an adversary retrieve from the ciphertext.

Hence, let us assume that there is an equivalence relation, $\equiv_{\boldsymbol{C}}$ given on the set of valid encryption terms, with the property that for any $M; N \in \boldsymbol{Enc_v}$ and $\boldsymbol{\sigma}$ key-renaming function, $\boldsymbol{M \equiv_C N}$ if and only if $\boldsymbol{M\sigma \equiv_C N\sigma}$. Let $\boldsymbol{Q_{Keys} := Enc_v/\equiv_C}$.

Since it must be required that $\boldsymbol{M \equiv_C N} \in \boldsymbol{Enc_v}$ if and only if $\boldsymbol{M\sigma \equiv_C N\sigma}$ whenever $\boldsymbol{\sigma}$ is a key-renaming function, $\boldsymbol{\sigma}$ induces a renaming on Q_{Enc}, which also is denoted by $\boldsymbol{\sigma}$.

Example 9.6 (Length-Revealing). Two encryption terms were considered to be indistinguishable for an adversary if and only if they

had the same length. In this case, let us define $\equiv_{\boldsymbol{C}}$ so that it equates encryption terms with the same length, and hence an element of Q_{Enc} will contain all encryption terms that have a specific length.

Example 9.7 (Which-Key Revealing). We have already considered the situation when an adversary can recognise that two encryption terms were encrypted with different keys. For this case, we will need to define $\equiv_{\boldsymbol{C}}$ so that two encryption terms are equivalent if and only if they are encrypted with the same key.

Definition 9.33 (Formal Logic of Symmetric Encryption). A formal logic for symmetric en-cryption is a triple $\Delta = (\boldsymbol{Exp_v}; \equiv_{\boldsymbol{K}}; \equiv_{\boldsymbol{C}})$ where Exp_V is a set of valid expressions, $\equiv_{\boldsymbol{K}}$ is an equivalence relation on Keys, and $\equiv_{\boldsymbol{C}}$ is an equivalence relation on Enc_V; we require the elements of Q_{Keys} to be infinite sets, and that for any $\boldsymbol{\sigma}$ key renaming function relative to Q_{Keys},

(i) if $M \in \text{Exp}$, then $M \in \text{Exp}_V$ if and only if $M\boldsymbol{\sigma} \in \text{Exp}_V$;

(ii) if $M, N \in \text{Enc}_V$, then $M \equiv_{\boldsymbol{C}} N$ if and only if $M\boldsymbol{\sigma} \equiv_{\boldsymbol{C}} N\boldsymbol{\sigma}$;

(iii) replacing an encryption term within a valid expression with another equivalent valid en-cryption term results in a valid expression.

To define the equivalence of expressions, let us assign to each valid expression an element in the set of *patterns*, Pat, defined the following way:

Definition 9.34 (Pattern). We define the set of *patterns*, Pat, by the grammar:

$$\text{Pat} ::= \text{Keys} \mid \text{Blocks} \mid (\text{Pat; Pat}) \mid \{\boldsymbol{Pat}\}_{\boldsymbol{Keys}} \mid \square_{\boldsymbol{Q_{Enc}}}$$

The pattern of a valid expression M, denoted by *pattern*(M), is obtained from M by replacing each undecryptable term $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \sqsubseteq \boldsymbol{M}(\boldsymbol{K} \notin \mathbf{R} - \mathbf{Keys})$) by $\square_{\boldsymbol{\mu}(\{\boldsymbol{M'}\}_{\boldsymbol{K}})}$, where $\boldsymbol{\mu}(\{\boldsymbol{M'}\}_{\boldsymbol{K}}) \in Q_{\text{Enc}}$ denotes the equivalence class containing $\{\boldsymbol{M'}\}_{\boldsymbol{K}}$.

Definition 9.35 (Equivalence of Expressions). We say that two valid expressions M and N are *equivalent*, and denote it by $\boldsymbol{M} \cong \boldsymbol{N}$, if there exists a key-renaming function $\boldsymbol{\sigma}$ such that *pattern*(M) = *pattern*($N\boldsymbol{\sigma}$), where for any pattern Q, $Q\boldsymbol{\sigma}$ denotes the pattern obtained

by re-naming all the keys and the box-indexes (which are equivalence classes in Q_{Enc}) in Q with $\boldsymbol{\sigma}$.

Example 9.8. In the case when the elements of Q_{Enc} contain encryption terms encrypted with the same key, Example 9.7, there is a one-to-one correspondence between Q_{Enc} and Keys, and therefore we can index the boxes with keys instead of the elements in Q_{Enc}: $\square_{\boldsymbol{K}}$, $K \in$ Keys. Then if N is the same expression as in Example 9.3, the pattern according to the above definition is the same as we had in that example. In that example M and N are equivalent according to the definition of equivalence above.

Proper Equivalence of Ciphers

In order to make the soundness and completeness proofs work, we need to have some restrictions on $\equiv_{\boldsymbol{C}}$; without any restrictions, the proofs will never work. The condition that we found the most natural for our purposes is what we call *proper equivalence*, defined below. This condition will make soundness work. For completeness, besides proper equivalence, we need to assume something for the relationship of $\equiv_{\boldsymbol{C}}$ and $\equiv_{\boldsymbol{K}}$. We call our assumption *independence*, and it is defined in Definition 2.37. Let us start by defining the set $\boldsymbol{\mu_{key}}$, for each $\boldsymbol{\mu} \in Q_{\text{Enc}}$, as

$$\boldsymbol{\mu_{key}} := \left\{ \begin{array}{c} \boldsymbol{K} \in \textbf{Keys} \mid \textbf{there is a valid expression } \boldsymbol{M} \\ \textbf{such that } \{\mathbf{M}\}_{\mathbf{K}} \in \boldsymbol{\mu} \end{array} \right\}.$$

Definition 9.36 (Proper Equivalence of Ciphers). We say that an equivalence relation $\equiv_{\boldsymbol{C}}$ on Enc_V is *proper*, if for any finite set of keys S, if $\boldsymbol{\mu} \in Q_{\text{Enc}}$ contains an element of the form $\{\boldsymbol{N}\}_{\boldsymbol{K}}$ with $K \notin S$, we have that:

- if $|\boldsymbol{\mu_{key}}|$ is finite then $\boldsymbol{\mu}$ also contains an element C such that $\mathit{Keys}(C) \cap S = \emptyset$, and $K \not\sqsubseteq C$;
- if $|\boldsymbol{\mu_{key}}| = \infty$ then $\boldsymbol{\mu}$ also contains an element C such that $\mathit{Keys}(C) \cap (S \cup \{\mathbf{K}\}) = \emptyset$.

In other words, if $\boldsymbol{\mu}$ contains an element encrypted with a key K not in S, then $\boldsymbol{\mu}$ has a representative in which no key of S appears, and in which K may only appear as an encrypting key, but not as a subexpression, or in the case of a class with infinitely many encrypting keys there is an element in which no keys from $S \cup \{\mathbf{K}\}$ appear. In fact, it was shown that the cardinality of the set $\boldsymbol{\mu}_{\boldsymbol{key}}$ is equal to either 1 or ∞.

Example 9.9. If $\equiv_{\boldsymbol{C}}$ denotes the equivalence of Example 2.7 (*i.e.* two ciphers are equivalent if they have the same encrypting key, hence $\left|\boldsymbol{\mu}_{\boldsymbol{key}}\right| = 1$), then it is clearly proper, since if $\{\boldsymbol{M}\}_{\boldsymbol{K}} \in \boldsymbol{\mu}$, and $K \notin S$, then $C = \{\boldsymbol{K'}\}_{\boldsymbol{K}}$ works for any $K' \notin S$; there is such a K', since we assumed that there are infinitely many keys. $C = \{\boldsymbol{B}\}_{\boldsymbol{K}}$ ($B \in$ Blocks) is also a good choice since Blocks is not empty.

Example 9.10. If $\equiv_{\boldsymbol{C}}$ denotes the equivalence of Example 2.6, then it is clearly proper ($\left|\boldsymbol{\mu}_{\boldsymbol{key}}\right|=\infty$). If $\{\boldsymbol{M}\}_{\boldsymbol{K}} \in \boldsymbol{\mu}$, $K \notin S$, then $C = \{\boldsymbol{M'}\}_{\boldsymbol{K'}}$ is a good choice where C is constructed by assigning to each key in $\{\boldsymbol{M}\}_{\boldsymbol{K}}$, a new key K'' not in $S \cup \{\mathbf{K}\}$. We can do this since we assumed that there are infinitely many keys. Then, since key-renaming does not change the length, $l(M) = l(M')$, and $\boldsymbol{\mu}$ contains all encryption terms of the same length, $C \in \boldsymbol{\mu}$ and properness follows.

The following propositions will be useful for proving our general soundness and complete-ness results.

Proposition 9.1. *Let* $\Delta = (Exp_V; \equiv_{\boldsymbol{K}}; \equiv_{\boldsymbol{C}})$ *be such that* $\equiv_{\boldsymbol{C}}$ *is proper. Then, the equivalence relation* $\equiv_{\boldsymbol{C}}$ *is such that for any equivalence class* $\boldsymbol{\mu} \in Q_{Enc}$, $\boldsymbol{\mu}_{\text{key}}$ *has either one, or infinitey many elements.*

Proof. Let $\boldsymbol{\mu} \in Q_{\text{Enc}}$, and assume that there are more than one encrypting key in $\boldsymbol{\mu}_{\text{key}}$ (but $\left|\boldsymbol{\mu}_{\boldsymbol{key}}\right|$ finite), that is, there are two different keys K and K_1 such that $\{\boldsymbol{M}\}_{\boldsymbol{K}}, \{\boldsymbol{M_1}\}_{\boldsymbol{K_1}} \in \boldsymbol{\mu}$ for some valid expressions M and M_1. Since $\equiv_{\boldsymbol{C}}$) is proper and $\{\boldsymbol{M_1}\}_{\boldsymbol{K_1}} \in \boldsymbol{\mu}$, if we consider $S = \{\boldsymbol{K}\}(\boldsymbol{K_1} \neq \boldsymbol{K}$ **thus** $\boldsymbol{K_1} \notin \boldsymbol{S})$ then $\boldsymbol{\mu}$ has an element of the form $\{\boldsymbol{M'}\}_{\boldsymbol{K'}}$

in which no key of *S* appears and in which K_1 may only appear as an encrypting key, but not as a subexpression. In particular we have that

$$\boldsymbol{K \notin Keys(M')} \boldsymbol{and} \boldsymbol{K \neq K'}$$

Since we assumed that each equivalence class in Q_{Keys} contains infinitely many elements (recall Definition 2.33), there is a key $L \neq K$ such that $L \equiv_K K$, and $\boldsymbol{L \notin \mathbf{Keys}(\{\mathbf{M}\}_{\mathbf{K}}) \cup \mathbf{Keys}(\{\mathbf{M}'\}_{\mathbf{K}'})}$.

Then, defining $\boldsymbol{\sigma}$ to do nothing else but to switch the keys *L* and *K*, we have using (2.2) that

$$\{M\}_{K^{\sigma}} = \{\boldsymbol{M\sigma}\}_{\boldsymbol{L}}$$

and (by (2.1) and (2.2)) $\{\boldsymbol{M'}\}_{\boldsymbol{K'}^{\sigma}} = \{\boldsymbol{M'}\}_{\boldsymbol{K'}}$

But, since $\{\boldsymbol{M}\}_{\boldsymbol{K}} \equiv_{\boldsymbol{C}} \{\boldsymbol{M'}\}_{\boldsymbol{K'}}$, we have (by definition of formal logic) that

$$\{\boldsymbol{M}\}_{\boldsymbol{K}^{\sigma}} \equiv_{\boldsymbol{C}} \{\boldsymbol{M'}\}_{\boldsymbol{K'^{\sigma}}}$$

that is

$$\{\boldsymbol{M\sigma}\}_{\boldsymbol{L}} \equiv_{\boldsymbol{C}} \{\boldsymbol{M'}\}_{\boldsymbol{K'}}$$

Since $\{\boldsymbol{M'}\}_{\boldsymbol{K'}} \in \boldsymbol{\mu}$, it must hold that $\{\boldsymbol{M\sigma}\}_{\boldsymbol{L}} \in \boldsymbol{\mu}$. Therefore, there are infinitely many encrypting keys in $\boldsymbol{\mu}$ since there are infinitely many choices for L.

Proposition 9.2. *Let* $\Delta = (\boldsymbol{Exp_v}, \equiv_{\boldsymbol{K}}, \equiv_{\boldsymbol{C}})$ *be such that* $\equiv_{\boldsymbol{C}}$ *is proper. If* $\boldsymbol{\sigma}$ *is a key-renaming function (relative to* $\equiv_{\boldsymbol{K}}$*), then for any* $\boldsymbol{\mu} \in \boldsymbol{Q_{Enc}}$, $\boldsymbol{|\mu_{key}| = |\sigma(\mu)_{key}|}$.

Proof. If $\boldsymbol{|\mu_{key}| = \infty}$, then $\boldsymbol{|\sigma(\mu)_{key}| = \infty}$, since for any $\{\boldsymbol{M}\}_{\boldsymbol{K}} \in \boldsymbol{\mu}$, $\{\boldsymbol{M}\}_{\boldsymbol{K}^{\sigma}} = \{\boldsymbol{M\sigma}\}_{\boldsymbol{\sigma(K)}} \in \boldsymbol{\sigma(\mu)}$. Since $\boldsymbol{\sigma}$ is a bijection, and since any $\boldsymbol{\mu}$ contains either only one or infinitely many elements, the claim follows.

The meaning of the next proposition is that if $\equiv_{\boldsymbol{C}}$ is proper, then given a set of valid ciphers $\boldsymbol{C = \{\{N_i\}_{L_i}\}_{i=1}^{n}}$ such that none of the encrypting keys are in *S*, and if $\boldsymbol{\mu_1, \ldots, \mu_l}$ are all the equivalence classes of the elements in C, then it is possible to choose a representative of each of $\boldsymbol{\mu_j}$, denoted by $\boldsymbol{C_{\mu j}}$, such that no key of S occurs in any of $\boldsymbol{C_{\mu j}}$, none of the L_i's occur as a subexpression in any $\boldsymbol{C_{\mu j}}$, and no key occurs in two of $\boldsymbol{C_{\mu j}}$unless the corresponding two equivalence classes both have only the same, single encrypting key.

Proposition 9.3. *Let* $\Delta = (\boldsymbol{Exp}_v, \equiv_K, \equiv_C))$ *be such that* $\equiv_C$ *is proper. Let* $\boldsymbol{C} = \{\{\boldsymbol{N_i}\}_{L_i}\}_{i=1}^{n}$ *be a set of valid encryption terms, and S a finite set of keys with* $\boldsymbol{L_i} \notin \mathbf{S}(\mathbf{i} \in \{\mathbf{1}, \ldots, \mathbf{n}\})$. *Let* $\boldsymbol{\mu}(\mathfrak{C})$ *denote the set of all equivalence-classes with respect to* $\equiv_C$ *of all elements in* $\mathfrak{C}$. *Then, for each* $\boldsymbol{v} \in \boldsymbol{\mu}(\mathfrak{C})$, *there is an element* $\boldsymbol{C_v} \in \boldsymbol{v}$ *such that:*

98. $\text{Keys}(\boldsymbol{C_v}) \cap \mathbf{S} = \emptyset$
99. $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_v}$ for all $\boldsymbol{i} \in \{\mathbf{1}, \ldots, \boldsymbol{n}\}$
100. $\boldsymbol{if}\ \boldsymbol{v} \neq \boldsymbol{v'} |\boldsymbol{v_{key}}| \neq \infty\ \boldsymbol{and}\ |\boldsymbol{v'_{key}}| \neq \infty$, then $\text{Keys}(\boldsymbol{C_v}) \cap \text{Keys}(\boldsymbol{C_{v'}}) = \emptyset$; if and only if $\boldsymbol{v_{key}} = \boldsymbol{v'_{key}}\ \{\boldsymbol{K}\}$ for some key K, and in this case:
    - $\text{Keys}(\boldsymbol{C_v}) \cap \text{Keys}(\boldsymbol{C_{v'}}) = \{\boldsymbol{K}\}$,
    - $\boldsymbol{C_v}$ and $\boldsymbol{C_{v'}}$ are both of the form $\{\bullet\}_{\boldsymbol{K}}$ with the same K, and
    - $\text{K} \not\sqsubseteq \boldsymbol{C_v}$, $\text{K} \not\sqsubseteq \boldsymbol{C_{v'}}$.
101. if $\boldsymbol{v} \neq \boldsymbol{v'}$ and either $|\boldsymbol{v_{key}}| = \infty$ or $|\boldsymbol{v'_{key}}| = \infty$, then $\text{Keys}(\boldsymbol{C_v}) \cap \text{Keys}(\boldsymbol{C_{v'}})) = \emptyset$.

Proof. Observe, that if $\boldsymbol{\mu_i}$ denotes the equivalence class of $\{\boldsymbol{N_i}\}_{L_i}$ in Q_{Enc}, then $v \in \boldsymbol{\mu}(\boldsymbol{C})$ if and only if $\text{v} = \boldsymbol{\mu_i}$ for some $\in \{\mathbf{1}, \ldots \boldsymbol{n}\}$. Proof goes by induction.

The statement is clearly true if $n = 1$, since $\equiv_C$ is proper.

Suppose now that the result is true for n - 1. Let $\{\boldsymbol{N_1}\}_{\boldsymbol{L_1}}$, $\{\boldsymbol{N_2}\}_{\boldsymbol{L_2}}$,..., $\{\boldsymbol{N_n}\}_{\boldsymbol{L_n}}$ be valid expressions, and let S be a set of keys such that $L_i \notin S$. Without loss of generality, we can assume, that the numbering is such that there is an $, \mathbf{1} \leq \boldsymbol{l} \leq \boldsymbol{n}$, such that $|(\boldsymbol{\mu_i})_{\boldsymbol{key}}|$ if $\boldsymbol{i} \leq \boldsymbol{l}$ and $|(\boldsymbol{\mu_i})_{\boldsymbol{key}}| = \infty\ \boldsymbol{if}\ > \boldsymbol{l}$.

Case 9.1: Let us first assume that $l = n$, i.e., $|(\boldsymbol{\mu_i})_{\boldsymbol{key}}| = \mathbf{1}$ for all $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n}$, and that there is an $\boldsymbol{m} \in \{\mathbf{1}, \ldots, \boldsymbol{n} - \mathbf{1}\}$ such that $L_n = L_m$. Since the statement is assumed to be true for *n-1* , we have that for the

family of encryption terms $\mathfrak{C}'$ $\{\{N_i\}_{L_i}\}_{i=1}^{n-1}$ and the set *S* we can choose C_{μ_i} for all $i \leq n-1$ such that conditions (i'), (ii'), (iii') and (iv') hold for these , that is,

(i') *Keys*(C_{μ_i}) ∩ *S* = ∅*;* for all $1 \leq i, j \leq n-1$,

(ii') $L_i \not\sqsubseteq C_{\mu_j}$ for all $1 \leq i, j \leq n-1$, and

(iii') if $\mu_i \neq \mu_j$, $|(\mu_i)_{key}| \neq \infty$ and $|(\mu_j)_{key}| \neq \infty$, then $Keys(C_{\mu_i}) \cap Keys(C_{\mu_j}) \neq \emptyset$ *;* if and only if $(\mu_i)_{key} = (\mu_j)_{key} = \{K\}$ for some key *K*, and in that case

102. $Keys(C_{\mu_i}) \cap Keys(C_{\mu_j}) = \{K\}$,

103. C_{μ_i} and C_{μ_j} are both of the form $\{\bullet\}_K$ with the same K, and

104. $K \not\sqsubseteq C_{\mu_i}$, $K \not\sqsubseteq C_{\mu_j}$.

(iv') if $\mu_i \neq \mu_j$ and either $|(\mu_i)_{key}| = \infty$ or $|(\mu_j)_{key}| = \infty$, then $Keys(C_{\mu_i}) \cap Keys(C_{\mu_j}) = \emptyset$.

We can immediately discard (iv') since we suppose that $|(\mu_i)_{key}| = 1$ for all $1 \leq i \leq n$. Suppose now that $\mu_n = \mu_j$ for some $i \leq n-1$, then there is nothing to prove, $C_{\mu_n} = C_{\mu_i}$ has already been chosen and so (i), (ii) and (iii) are obviously satisfied by IH.

If there is no such *i*, then consider

$$S_{n-1} := (\left(\bigcup_{i=1}^{n-1} Keys(C_{\mu_i}) \cup \{L_i\}\right) \backslash \{L_n\}) \cup S$$

Since $\equiv_C$ is proper (using S_{n-1} and $\{N_n\}_{L_n} \in \mu_n$), there is a $C \in \mu_n$ such that $Keys(C) \cup S_{n-1} = \emptyset$ and $L_n \not\sqsubseteq C$. Let us define $C_{\mu_n} = C$. Then:

(i) $Keys(C) \cap S = \emptyset$ *;* follows from the fact that $Keys(C) \cap S_{n-1} =$ *;* and $S \subseteq S_{n-1}$;

(ii) $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$ for all $\mathbf{1} \leq \boldsymbol{i,j} \leq \boldsymbol{n}$ since:

1. $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, for all $\mathbf{1} \leq \boldsymbol{i,j} \leq \boldsymbol{n-1}$ by (ii'),
2. $\boldsymbol{L_n} \not\sqsubseteq C_{\mu_j}$, $\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n-1}$ because we assumed that $L_n = L_m$ and $\boldsymbol{L_m} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$ by (ii'),
3. $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C}$, for all $\boldsymbol{L_i} \neq \boldsymbol{L_m}$ such that $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n-1}$ (remember that $L_n = L_m$) since $\boldsymbol{L_i} \in \boldsymbol{S_{n-1}}$ and $\boldsymbol{Keys(C)} \cap \boldsymbol{S_{n-1}} = \emptyset$, and
4. $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C}$ by the way that C was chosen (hence $\boldsymbol{L_m} \not\sqsubseteq \boldsymbol{C}$).

(iii)

1. for all $\mathbf{1} \leq \boldsymbol{i,j} \leq \boldsymbol{n-1}$ it is true by (iii');
2. Suppose now that $\boldsymbol{\mu_n} \neq \boldsymbol{\mu_k}$ and $(\boldsymbol{C}) \cap \boldsymbol{Keys}(\boldsymbol{C_{\mu_k}}) \neq \emptyset$; for some $\mathbf{1} \leq \boldsymbol{k} \leq \boldsymbol{n-1}$. If we combine these with the fact that $\boldsymbol{Keys(C)} \cap \boldsymbol{S_{n-1}} = \emptyset$, we need to have that
$$\boldsymbol{Keys(C)} \cap \boldsymbol{Keys}(\boldsymbol{C_{\mu_k}}) = \{\boldsymbol{L_n}\}.$$

It is now easy to see from the equation above that *C* and $\boldsymbol{C_{\mu_k}}$ are both of the form $\{\bullet\}_{\boldsymbol{L_n}}$. For that notice that by (ii.d) just proved above, $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C}$ and by (ii.a) $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_k}}$. The only thing left to show is that $(\boldsymbol{\mu_n})_{\boldsymbol{key}} = (\boldsymbol{\mu_k})_{\boldsymbol{key}} = \{\boldsymbol{L_n}\}$. This comes straightforward from the fact that *C* and $\boldsymbol{C_{\mu_k}}$are both of the form $\{\bullet\}_{\boldsymbol{L_n}}$ and from the fact that $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \mathbf{1}$ for all $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n}$. Combining these we have
$$(\boldsymbol{\mu_n})_{\boldsymbol{key}} = (\boldsymbol{\mu_k})_{\boldsymbol{key}} = \{\boldsymbol{L_n}\}.$$

The converse is very simple. Suppose that $(\boldsymbol{\mu_n})_{\boldsymbol{key}} = (\boldsymbol{\mu_k})_{\boldsymbol{key}} = \{\boldsymbol{L_n}\}$. Since $\boldsymbol{C} \in \boldsymbol{\mu_n}$ and $\boldsymbol{C_{\mu_k}} \in \boldsymbol{\mu_k}$ we have that both are of the form $\{\bullet\}_{\boldsymbol{L_n}}$ and thus $\boldsymbol{Keys}(\boldsymbol{C}) \cap \boldsymbol{Keys}(\boldsymbol{C_{\mu_k}}) \neq \emptyset$. The rest follows as above.

(iv) Verified since by hypothesis we suppose that $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \mathbf{1}$ for all $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n}$.

Case 9.2: Suppose now that *l* = *n*, but there is no $\boldsymbol{m} \in \{\mathbf{1}, \ldots, \boldsymbol{n}-\mathbf{1}\}$ such that $L_n = L_m$. Since the result is true for *n-1*, we have that for the family of encryption terms $\mathfrak{C}'\{\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}\}_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}-\mathbf{1}}$ and the set $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{L_n}\}$ (note that $\boldsymbol{L_i} \notin \boldsymbol{S'}$ for all $\boldsymbol{i} \leq \boldsymbol{n}-\mathbf{1}$) we can choose $\boldsymbol{C_{\mu_i}}$ for all $\boldsymbol{i} \leq \boldsymbol{n}-\mathbf{1}$ such that conditions (i'), (ii'), (iii') and (iv') hold for these , that is,

(i') $Keys(\boldsymbol{C_{\mu_i}}) \cap (\boldsymbol{S} \cup \{\boldsymbol{L_n}\}) = \emptyset$ for all $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n}-\mathbf{1}$,

(ii') $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$ for all $\mathbf{1} \leq \boldsymbol{i}, \boldsymbol{j} \leq \boldsymbol{n}-\mathbf{1}$, and

(iii') if $\boldsymbol{\mu_i} \neq \boldsymbol{\mu_j}$, $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| \neq \infty$ and $\left|(\boldsymbol{\mu_j})_{\boldsymbol{key}}\right| \neq \infty$, then $\boldsymbol{Keys}(\boldsymbol{C_{\mu_i}}) \cap \boldsymbol{Keys}(\boldsymbol{C_{\mu_j}}) \neq \emptyset$ if and only if $(\boldsymbol{\mu_i})_{\boldsymbol{key}} = (\boldsymbol{\mu_j})_{\boldsymbol{key}} = \{\boldsymbol{K}\}$ for some key *K*;

(iv') if $\boldsymbol{\mu_i} \neq \boldsymbol{\mu_j}$ and either $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \infty$ or $\left|(\boldsymbol{\mu_j})_{\boldsymbol{key}}\right| = \infty$, then $\boldsymbol{Keys}(\boldsymbol{C_{\mu_i}}) \cap \boldsymbol{Keys}(\boldsymbol{C_{\mu_j}}) = \emptyset$.

Again, if $\boldsymbol{\mu_n} = \boldsymbol{\mu_i}$ for some *i* < *n*, then there is nothing to prove, let $\boldsymbol{C_{\mu_n}} = \boldsymbol{C_{\mu_i}}$ and note that (i) and (iii) are obviously satisfied, and (ii) ($\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, for all $\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n}$, and $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_n}}$ for all $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n}-\mathbf{1}$) follows from (i') and (ii') respectively. Again (iv) is also true since we suppose that $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \mathbf{1}$ for all $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{n}$.

If there is no such *i*, then consider

$$\boldsymbol{S_{n-1}}\left(\bigcup_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}-\mathbf{1}} \boldsymbol{Keys}(\boldsymbol{C_{\mu_i}}) \cup \{\boldsymbol{L_i}\}\right) \cup \mathbf{S}.$$

By properness (using $\boldsymbol{S_{n-1}}$ and $\{\boldsymbol{N_n}\}_{\boldsymbol{L_n}} \in \boldsymbol{\mu_n}$), and since $\boldsymbol{L_n} \notin \boldsymbol{S_{n-1}}$ (by (i') assumption $\boldsymbol{L_n} \neq \boldsymbol{L_i}$ for all *i* < *n*, and by hypothesis of the proposition $\boldsymbol{L_n} \notin \mathbf{S}$), there is a $\boldsymbol{C} \in \boldsymbol{\mu_n}$ such that $\boldsymbol{Keys}(\boldsymbol{C}) \cap \boldsymbol{S_{n-1}} = \emptyset$, and $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C}$. Let us define $\boldsymbol{C_{\mu_n}} = \boldsymbol{C}$. Then:

(i)follows from (i') and from the fact that $\boldsymbol{Keys}(\boldsymbol{C}) \cap \boldsymbol{S_{n-1}} = \emptyset$;

(ii) is true, since:

1. $L_i \not\sqsubseteq C_{\mu_j}$, for all $1 \leq i, j \leq n-1$ by (ii'),
2. $L_n \not\sqsubseteq C_{\mu_j}$, for all $1 \leq j \leq n-1$ by (i'),
3. $L_i \not\sqsubseteq C$ for $1 \leq i \leq n-1$ because by properness $Keys(C) \cap S_{n-1} = \emptyset$, and
4. $L_n \not\sqsubseteq C$ because of properness.

(iii) follows, because:

1. for all $1 \leq i, j \leq n-1$ it is true by (iii'), and
2. for the other case it holds since by properness $Keys(C) \cap S_{n-1} = \emptyset$ and thus $Keys(C) \cap Keys(C_{\mu_i}) = \emptyset$ for all $i \leq i \leq n-1$.

(iv)Verified since by hypothesis we suppose that $|(\mu_i)_{key}| = 1$ for all $1 \leq i \leq n$.

Case 9.3: Suppose now that $l < n$, but there is $m \in \{1, \ldots, n-1\}$ such that $L_n = L_m$. Since the result is assumed to be true for n - 1, we have that for the family of encryption terms $\mathfrak{C}' = \{\{N_i\}_{L_i}\}_{i=1}^{n-1}$ and the set S we can choose C_{μ_i} for all $i \leq n-1$ such that conditions (i'), (ii'), (iii') and (iv') hold for these , that is,

(i') Keys(C_{μ_i}) ∩ S = ∅ for all $1 \leq i \leq n-1$,

(ii') $L_i \not\sqsubseteq C_{\mu_j}$ for all $1 \leq i, j \leq n-1$, and

(iii') if $\mu_i \neq \mu_j, |(\mu_i)_{key}| \neq \infty$ and $|(\mu_j)_{key}| \neq \infty$, then Keys($C_{\mu_i}$) ∩ Keys($C_{\mu_j}$) ≠ ∅ if and only if $(\mu_i)_{key} = (\mu_j)_{key} = \{K\}$ for some key K;

(iv') if $\mu_i \neq \mu_j$ and either $|(\mu_i)_{key}| = \infty$ or $|(\mu_j)_{key}| = \infty$, then Keys(C_{μ_i}) ∩ Keys(C_{μ_j}) = ∅.

Again, suppose now that $\boldsymbol{\mu_n = \mu_i}$ for some $\boldsymbol{i \leq n-1}$, then there is nothing to prove, $\boldsymbol{C_{\mu_n}}$ =has already been chosen and so (i), (ii), (iii) and (iv) are obviously satisfied by IH.

If there is no such *i*, then consider

$$\boldsymbol{S_{n-1} := ((\bigcup_{i=1}^{n-1} Keys(C_{\mu_i}) \cup \{L_i\}) \backslash \{L_n\}) \cup S}$$

Since $\equiv_{\boldsymbol{C}}$ is proper (using $\boldsymbol{S_{n-1}}$ and $\{\boldsymbol{N_n}\}_{\boldsymbol{L_n}} \in \boldsymbol{\mu_n}$, $\left|(\boldsymbol{\mu_n})_{\boldsymbol{key}}\right| = \infty$), there is a $\boldsymbol{C} \in \boldsymbol{\mu_n}$ such that *Keys*(*C*) ∩ ($\boldsymbol{S_{n-1}} \cup \{\boldsymbol{L_n}\}$) = ∅. Then:

(i) *Keys*(*C*) ∩ $\boldsymbol{S}$ = ∅ follows from the fact that *Keys*(*C*)∩ ($\boldsymbol{S_{n-1}} \cup \{\boldsymbol{L_n}\}$) = ∅ and *S* ⊆ ($\boldsymbol{S_{n-1}} \cup \{\boldsymbol{L_n}\}$);

(ii) $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$ for all $\boldsymbol{1 \leq i,j \leq n}$ since:

1. $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, for all $\boldsymbol{1 \leq i,j \leq n-1}$ by (ii'),
2. $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, $\boldsymbol{1 \leq j \leq n-1}$ because we assumed that $L_n = L_m$ and $L_m \not\sqsubseteq \boldsymbol{C_{\mu_j}}$ by (ii'),
3. $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C}$, for all $L_i \neq \boldsymbol{L_m}$ such that $\boldsymbol{1 \leq i \leq n-1}$ (remember that $L_n = L_m$) since $\boldsymbol{L_i} \in \boldsymbol{S_{n-1}}$ and Keys(C)∩ $\boldsymbol{S_{n-1}}$ = ∅, and
4. $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C}$ because Keys(C) ∩ ($\boldsymbol{S_{n-1}} \cup \{\boldsymbol{L_n}\}$) = ∅ (hence $\boldsymbol{L_m} \not\sqsubseteq \boldsymbol{C}$).

(iii) note that if $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| \neq \infty$ ***and*** $\left|(\boldsymbol{\mu_j})_{\boldsymbol{key}}\right| \neq \infty$ then $\boldsymbol{1 \leq i,j \leq l < n}$ and thus by HI (iii) holds.

(iv)

1. for the case $\boldsymbol{l \leq j \leq n-1}$ and $\boldsymbol{1 \leq i \leq n-1}$, Keys($\boldsymbol{C_{\mu_j}}$) ∩ Keys($\boldsymbol{C_{\mu_i}}$)= ∅ holds by IH;
2. it is only left to show that for all $\boldsymbol{1 \leq i \leq n-1}$, Keys($\boldsymbol{C_{\mu_n}}$) ∩Keys($\boldsymbol{C_{\mu_i}}$) = ∅. This is true because by

definition Keys($\boldsymbol{C_{\mu_n}}$) ∩ ($\boldsymbol{S_{n-1}}$ ∪ {$\boldsymbol{L_n}$}) = ∅ and Keys($\boldsymbol{C_{\mu_i}}$)) ⊆ ($\boldsymbol{S_{n-1}}$ ∪ {$\boldsymbol{L_n}$})).

Case 9.4: The proof of the remaining case, $l < n$, i.e., $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \infty$ for $l < i \leq n$, and there is no $\boldsymbol{m} \in \{\boldsymbol{1}, \ldots, \boldsymbol{n-1}\}$ such that $L_n = L_m$ is a combination of the proofs of Case 2 and Case 3. Since the result is true for $\boldsymbol{n-1}$, we have that for the family of encryption terms $\mathcal{C}' \left\{\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}\right\}_{\boldsymbol{i=1}}^{\boldsymbol{n-1}}$ and the set $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{L_n}\}$ (note that $\boldsymbol{L_i} \notin \boldsymbol{S'}$ for all $i \leq \boldsymbol{n-1}$) we can choose $\boldsymbol{C_{\mu_i}}$ for all $i \leq \boldsymbol{n-1}$ such that conditions (i'), (ii'), (iii') and (iv') hold for these , that is,

(i') *Keys*($\boldsymbol{C_{\mu_i}}$) ∩ ($\boldsymbol{S}$ ∪ {$\boldsymbol{L_n}$}) = ∅ for all $\boldsymbol{1 \leq i \leq n-1}$;

(ii') $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$ for all $\boldsymbol{1 \leq i, j \leq n-1}$, and

(iii') if $\boldsymbol{\mu_i \neq \mu_j}$, $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| \neq \infty$ and $\left|(\boldsymbol{\mu_j})_{\boldsymbol{key}}\right| \neq \infty$, then *Keys*($\boldsymbol{C_{\mu_i}}$) ∩ *Keys*($\boldsymbol{C_{\mu_j}}$) ≠ ∅ if and only if $(\boldsymbol{\mu_i})_{\boldsymbol{key}} = (\boldsymbol{\mu_j})_{\boldsymbol{key}} = \{\boldsymbol{K}\}$ for some key *K*;

(iv') if $\boldsymbol{\mu_i \neq \mu_j}$ and either $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \infty$ or $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \infty$, then *Keys*($\boldsymbol{C_{\mu_i}}$) ∩ *Keys*($\boldsymbol{C_{\mu_j}}$) = ∅.

Again, if $\boldsymbol{\mu_n = \mu_i}$ for some $i < n$, then there is nothing to prove, let $\boldsymbol{C_{\mu_n} = C_{\mu_i}}$ and note that (i), (iii) and (iv) are obviously satisfied, and (ii) ($\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, for all $\boldsymbol{1 \leq j \leq n}$, and $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_n}}$ for all $\boldsymbol{1 \leq i \leq n-1}$) follows from (i') and (ii') respectively.

If there is no such *i*, then consider

$$\boldsymbol{S_{n-1}} := ((\bigcup_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \boldsymbol{Keys}(\boldsymbol{C_{\mu_i}}) \cup \{\boldsymbol{L_i}\}) \cup \boldsymbol{S}$$

By properness (using $\boldsymbol{S_{n-1}}$ and $\{\boldsymbol{N_n}\}_{\boldsymbol{L_n}} \in \boldsymbol{\mu_n}$, $\left|(\boldsymbol{\mu_n})_{\boldsymbol{key}}\right| = \infty$), and since $\boldsymbol{L_n} \notin \boldsymbol{S_{n-1}}$ (by (i'), assumption $\boldsymbol{L_n \neq L_i}$ for all $i < n$, and by hypothesis of the proposition $L_n \notin \boldsymbol{S}$), there is a $\boldsymbol{C} \in \boldsymbol{\mu_n}$ such that *Keys*(*C*) ∩ ($\boldsymbol{S_{n-1}}$ ∩ {$\boldsymbol{L_n}$}) = ∅. Let us define $\boldsymbol{C_{\mu_n}} = C$. Then:

(i) follows from (i') and from the fact that *Keys*(*C*) ∩ $\boldsymbol{S_{n-1}} = \emptyset$;
(ii) is true, since:

1. $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, for all $1 \leq \boldsymbol{i,j} \leq \boldsymbol{n-1}$ by (ii'),
2. $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C_{\mu_j}}$, for all $1 \leq \boldsymbol{j} \leq \boldsymbol{n-1}$ by (i'),
3. $\boldsymbol{L_i} \not\sqsubseteq \boldsymbol{C}$ for $1 \leq \boldsymbol{i} \leq \boldsymbol{n-1}$ because by properness Keys(C) ∩ $\boldsymbol{S_{n-1}} = \emptyset$, and
4. $\boldsymbol{L_n} \not\sqsubseteq \boldsymbol{C}$ because by definition of C, Keys(C) ∩ $\boldsymbol{S_{n-1}}$ ∪ $\{\boldsymbol{L_n}\} = \emptyset$.

(iii) note that if $\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| \neq \infty$), and $\left|(\boldsymbol{\mu_j})_{\boldsymbol{key}}\right| \neq \infty$), then $1 \leq \boldsymbol{i,j} \leq \boldsymbol{l} < \boldsymbol{n}$ and thus by HI (iii) holds.
(iv)

1. for the case $1 \leq \boldsymbol{j} \leq \boldsymbol{n-1}$ and $1 \leq \boldsymbol{i} \leq \boldsymbol{n-1}$, Keys($\boldsymbol{C_{\mu_j}}$) ∩ Keys($\boldsymbol{C_{\mu_i}}$) = ∅ holds by IH;
2. it is only left to show that for all $1 \leq \boldsymbol{i} \leq \boldsymbol{n-1}$, Keys($\boldsymbol{C_{\mu_n}}$) ∩Keys($\boldsymbol{C_{\mu_i}}$) = ∅. This is true because by definition Keys($\boldsymbol{C_{\mu_n}}$) ∩ ($\boldsymbol{S_{n-1}}$ ∪ $\{\boldsymbol{L_n}\}$) = ∅ and Keys($\boldsymbol{C_{\mu_i}}$) ⊆ $\boldsymbol{S_{n-1}}$.

Given sets C and s as iin the conditions og the proposition, let R(C,S) denote the nonempty set

$$R(C,S) := \left\{ \left\{ \{C_v\}_{v \in \mu(C)} \mid C_v \in v, \text{and} \{C_v\}_{v \in C} \text{and} \quad S \quad \text{satisfy} \quad \text{conditions} \right\} \right\}$$

Another useful property satisfied by all common logics, and that we will need for the completeness result is the following:

Definition 9.37 (Independent $\equiv_{\boldsymbol{K}}$ and $\equiv_{\boldsymbol{C}}$). We say that $\equiv_{\boldsymbol{K}}$ and $\equiv_{\boldsymbol{C}}$ are independent, if for any finite set of keys *S*, and any finite set of ciphers C such that no key in *S* appears in any element of C, given any key-renaming function $\boldsymbol{\sigma}$, there is a key renaming $\boldsymbol{\sigma}$' for which $\boldsymbol{\sigma}$'(*K*) = *K* whenever $\boldsymbol{K} \in \boldsymbol{S}$, and for all *C* ∈ C, $C\,\boldsymbol{\sigma} \equiv_{\boldsymbol{C}} C\,\boldsymbol{\sigma}$'.

In other words, $\equiv_{\boldsymbol{K}}$ and $\equiv_{\boldsymbol{C}}$ are independent, if for any finite set of keys *S*, and any finite set of ciphers C such that no key in *S* appears in any element of C, it is possible to alter any key-renaming function $\boldsymbol{\sigma}$ such that the altered function leaves all the elements in *S* unchanged, whereas on C it does the same thing as the original $\boldsymbol{\sigma}$. We will need this property for the general completeness theorem.

9.5.3 **Interpretation**

The idea of the interpretation is to describe messages that are built from blocks of strings and keys via pairing and encryption. To each valid formal expression *M*, the interpretation assigns a random variable Φ(*M*) taking values in $\overline{\boldsymbol{strings}}$. We do not give one specific interpreting function though, we will just say that a function Φ is an interpretation if it satisfies certain properties. Let us assume, that a function ∅ is fixed in advance, which assigns to each formal key a key-generation algorithm. If Φ (*B*) ∈ $\overline{\boldsymbol{strings}}$ (constant random variable) is given for blocks, then, the rest of Φ is determined the following way: First, run the key-generation algorithm assigned by ∅ for each key in *Keys*(*M*). Then, using the outputs of these key-generations, translate the formal expressions according to the following rules: for each key, use the output of the corresponding key-generation. For blocks, just use Φ(*B*). For each pair, apply [• *;* •] to the interpretations of the expressions inside the formal pair. For each formal encryption, run the encryption algorithm using as key the bitstring that was output by the key generation, to encrypt the interpretation of the formal expression inside the formal encryption. The randomness of Φ (*M*) comes from the initial key-generation, and from running the encryption algorithm independently for each formal encryption. Let us define below this notion of interpretation with the following example:

Example 9.11. For $M = \left(\left(\{\mathbf{0}\}_{\boldsymbol{K}_{10}}, \boldsymbol{K}_{\mathbf{5}}\right), \{\boldsymbol{K}_{\mathbf{10}}\}_{\boldsymbol{K}_{\mathbf{5}}}\right)$, the interpretation is $\Phi(\mathrm{M}):(\boldsymbol{\Omega}_{\boldsymbol{E}} \times \boldsymbol{\Omega}_{\boldsymbol{E}}) \times \left(\boldsymbol{\Omega}_{\boldsymbol{\Phi}(\mathbf{K}_5)} \times \boldsymbol{\Omega}_{\boldsymbol{\Phi}(\mathbf{K}_{10})}\right) \rightarrow \overline{\boldsymbol{strings}}$, where $\boldsymbol{\Phi}(\boldsymbol{M})(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \boldsymbol{\omega}_4)$ is

$$\left[\begin{matrix}\left[E\left(\Phi(\mathbf{K_{10}})(\omega_4), \Phi(\mathbf{0})\right)(\omega_1), \Phi(\mathbf{K_5})(\omega_3)\right], E(\Phi(\mathbf{K_5})(\omega_3), \\ \Phi(\mathbf{K_{10}})(\omega_4))(\omega_2)\end{matrix}\right]$$

There are four instances of randomness, two coming from the generation of keys by the key-generation algorithm (for K_5 and for K_{10}), and the other two from the two encryptions ($\{\mathbf{0}\}_{K_{10}}$) and ($\{K_{10}\}_{K_5}$).

Definition 9.38 (Interpretation of Formal Expressions). Let $\mathbf{\Pi} = (\{\mathbf{K_i}\}_{i\in I}, \mathbf{E}, \mathbf{D}, \approx)$ be a general symmetric encryption scheme with some index set *I*, with $\{(\Omega_{K_i}, Pr_{K_i})\}_{i\in I}$ denoting the probability fields for key generation, and with (Ω_E, Pr_E) denoting the probability field for the randomness of encryption. Let Exp_V be a set of valid expressions. For each valid expression *M*, let the probability space (Ω_M, Pr_M) be defined recursively as

$$(\Omega_{\mathbf{K}}, \mathbf{Pr_K}) := (\{\omega_0\}, \mathbf{1}_{\{w_0\}}) \text{ for K} \in \text{Keys};$$
$$(\Omega_{\mathbf{B}}, \mathbf{Pr_B}) := (\{\omega_0\}, \mathbf{1}_{\{w_0\}}) \text{ for B} \in \text{Blocks};$$
$$(\Omega_{(\mathbf{M},\mathbf{N})}, \mathbf{Pr}_{(\mathbf{M},\mathbf{N})} := (\Omega_{\mathbf{M}} \times \Omega_{\mathbf{N}}, \mathbf{Pr_M} \otimes \mathbf{Pr_N});$$
$$(\Omega_{\{\mathbf{M}\}_{\mathbf{K}}}, \mathbf{Pr}_{\{\mathbf{M}\}_{\mathbf{K}}}) := (\Omega_{\mathbf{E}} \times \Omega_{\mathbf{M}}, \mathbf{Pr_E} \otimes \mathbf{Pr_M}).$$

Where $(\{\omega_0\}, \mathbf{1}_{\{w_0\}})$ is just the trivial probability-space with one elementary event, ω_0 only; the tensor product stands for the product probability. Suppose that a function $\emptyset : Keys \to \{K_i\}_{i\in I}$ is given assigning abstract keys to key generation algorithms, such that $\emptyset$(K) = $\emptyset(K')$and only if $K \equiv_K K'$. Let $i : \{1, \ldots, |Keys(M)|\} \to Keys(M)$ be a bijection enumerating the keys in *Keys*(*M*). Let

$$\left(\Omega_{Keys(M)}, Pr_{Keys(M)}\right) := \left(\begin{matrix}\Omega_{\Phi(\mathbf{i}(1))} \times \ldots \times \Omega_{\Phi(\mathbf{i}(|\mathbf{Keys(M)}|))}, Pr_{\Phi(\mathbf{i}(1))} \otimes \\ \ldots \otimes Pr_{\Phi(\mathbf{i}(|\mathbf{Keys(M)}|))}\end{matrix}\right)$$

The function $(M, M') \mapsto \left(\Phi_M(M') : \Omega_{M'} \times \Omega_{Keys(M)} \to \overline{strings}\right)$ defined whenever $M' \sqsubseteq M$, is ca1lled *an interpretation function*, if it satisfies the following properties:

$\Phi_M(B)\ (\omega_0, \omega) = \Phi_N(B)(\omega_0, \omega)$for all *M*, *N* valid expressions, *B* ∈ Blocks, $B \sqsubseteq M$,

$B \sqsubseteq N$, and arbitrary $\boldsymbol{\omega} \in \boldsymbol{\Omega_{Keys(M)}}, \boldsymbol{\omega'} \in \boldsymbol{\Omega_{Keys(N)}}$. Let $\boldsymbol{\Phi(B)} \coloneqq \boldsymbol{\Phi_M(B)}$.

$\boldsymbol{\Phi_M(K)}\left(\boldsymbol{\omega_0}, \left(\boldsymbol{\omega_1}, \dots, \boldsymbol{\omega}_{|\boldsymbol{Keys(M)}|}\right)\right) = \emptyset(\boldsymbol{K})\left(\boldsymbol{\omega}_{\boldsymbol{i}^{-1}(\boldsymbol{K})}\right)$ for $K \in Keys(M)$, with $\boldsymbol{\omega_j} \in \boldsymbol{\Omega}_{\emptyset(\boldsymbol{i}(\boldsymbol{j}))}$.

$\boldsymbol{\Phi_M}\left((\boldsymbol{M'}, \boldsymbol{M''})\right)\left((\boldsymbol{\omega'}, \boldsymbol{\omega''}), \boldsymbol{\omega}\right) =$
$[\boldsymbol{\Phi_M(M')}(\boldsymbol{\omega'}, \boldsymbol{\omega}), \boldsymbol{\Phi_M(M'')}(\boldsymbol{\omega''}, \boldsymbol{\omega})]$ for all $\boldsymbol{\omega'} \in \boldsymbol{\Omega_{M'}}, \boldsymbol{\omega''} \in \boldsymbol{\Omega_{M''}}$, and $\boldsymbol{\omega} \in \boldsymbol{\Omega_{Keys(M)}}$ if $(M', M'') \sqsubseteq M$.

$\boldsymbol{\Phi_M}(\{\boldsymbol{M'}\}_{\boldsymbol{K}})\left((\boldsymbol{\omega_E}, \boldsymbol{\omega'}), \boldsymbol{\omega}\right) =$
$\boldsymbol{E}\left(\boldsymbol{\Phi_M(K)}(\boldsymbol{\omega_0}, \boldsymbol{\omega})\boldsymbol{\Phi_M(M')}(\boldsymbol{\omega'}, \boldsymbol{\omega})\right)(\boldsymbol{\omega_E})$ for all $\boldsymbol{\omega_E} \in \boldsymbol{\Omega_E}$,

$\boldsymbol{\omega'} \in \boldsymbol{\Omega_{M'}}, \boldsymbol{\omega} \in \boldsymbol{\Omega_{Keys(M)}}$ if $\{\boldsymbol{M'}\}_{\boldsymbol{K}} \sqsubseteq \boldsymbol{M}$.

Let $\boldsymbol{\Phi}(M) := \boldsymbol{\Phi_M(M)}$, and let $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ denote the distribution of $\boldsymbol{\Phi}(M)$.

Soundness

An interpretation assigns a random variable $\boldsymbol{\Phi}(M)$ (and the distribution $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ of $\boldsymbol{\Phi}(M)$) to a formal valid expression M. On the set of valid expressions the equivalence $\cong$ equates expressions that a formal adversary supposedly cannot distinguish, whereas the equivalence $\approx$ equates random variables (and distributions) that a probabilistic adversary is not supposed to be able to distinguish. The question is, how the formal and the probabilistic equivalence are related through the interpretation. We say that soundness holds if $\boldsymbol{M} \cong \boldsymbol{N}$ implies $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$, whereas we say that completeness holds if $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$ implies $\boldsymbol{M} \cong \boldsymbol{N}$.

The key to a soundness theorem is to have enough boxes in the definition of formal equiva-lence, *i.e.*, there should be enough elements in Q_{Enc}. It is clear that in the extreme case, when the equivalence on encryption terms, $\equiv_{\boldsymbol{C}}$, is defined so that two encryption terms are equivalent iff they are the same, then soundness holds trivially for all interpretations; but this would be completely impractical, it would

assume a formal adversary that can see everything inside every encryption. It is also immediate, that if soundness holds with a given $\equiv_{\boldsymbol{C}}$ (and a given interpreta-tion), and $\equiv'_{\boldsymbol{C}}$ is such that for any to encryption terms *M* and *N*, $M \equiv'_{\boldsymbol{C}} N$ implies $M \equiv_{\boldsymbol{C}} N$ (i.e. $\equiv'_{\boldsymbol{C}}$has more boxes), then, keeping the same interpretation, soundness holds with the new $\equiv'_{\boldsymbol{C}}$as well. Hence, in a concrete situation, the aim is to introduce enough boxes to achieve soundness, but not too many, to sustain practicality. One way to avoid having too many boxes is to require completeness: we will see later, that obtaining completeness requires that we do not have too many boxes.

The following theorem claims the equivalence of two conditions. It is almost trivial that con-dition (i) implies condition (ii). The claim that (ii) implies (i) can be summarised the following way: if soundness holds for pairs of valid expressions *M* and *M′* with a special relation between them (described in (ii)), then soundness holds for all expressions (provided that they do not have encryption cycles). In other words, if $\boldsymbol{M} \cong \boldsymbol{M'}$ implies $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$ pairs *M* and *M′* then $\boldsymbol{M} \cong$ implies $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$ for certain specified.

Theorem 9.14. *Let* $\Delta = (\boldsymbol{Exp_v}, \equiv_{\boldsymbol{K}}, \equiv_{\boldsymbol{C}})$ *be a formal logic for symmetric encryption such* $\equiv_{\boldsymbol{C}}$ *is proper and for each* $M \in \boldsymbol{Exp_v}$*, B-Keys*(*M*) *is not cyclic in M. Let* $\pi = (\{\boldsymbol{K_i}\}_{\boldsymbol{i \in I}}, \boldsymbol{E}, \boldsymbol{D}, \approx)$ *be a general encryption scheme,* Φ *an interpretation of* $\boldsymbol{Exp_v}$ *in* π*. Then the following conditions are equivalent:*

(i) *Soundness holds for* Φ*:* $\boldsymbol{M} \cong \boldsymbol{N}$*, implies* $\boldsymbol{\Phi}(\boldsymbol{M}) \approx \boldsymbol{\Phi}(\mathbf{N})$;

(ii) *For any* $\boldsymbol{C} = \left\{\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}\right\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}$ *set of valid encryption terms, and S finite set of keys with* $L_i \notin S$, $(\boldsymbol{i} \in \{\boldsymbol{1}, \dots, \boldsymbol{n}\})$*, there is an element* $\{\boldsymbol{C_v}\}_{\boldsymbol{v} \in \boldsymbol{\mu}(\boldsymbol{C})}$ *of* R(C*; S*) *such that the followings hold:*

if $\left\{\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}}\right\}_{\boldsymbol{j=1}}^{\boldsymbol{l}} \subset C$ *and* $M \in \boldsymbol{Exp_v}$ *are such that*

1. $\left\{\boldsymbol{N_{i_1}}\right\}_{\boldsymbol{K}}, \left\{\boldsymbol{N_{i_2}}\right\}_{\boldsymbol{K}}, \dots \left\{\boldsymbol{N_{i_l}}\right\}_{\boldsymbol{K}} \sqsubseteq \boldsymbol{M}$,
2. R-Keys(M) ⊆ S, and

3. K does not occur anywhere else in M, all visible undecryptable encryption terms in M are elements of $\boldsymbol{C} \cup \{\boldsymbol{C_v}\}_{\boldsymbol{v} \in \boldsymbol{\mu}(\boldsymbol{C})}$, then, if we denote by M' the expression obtained by replacing in M each $\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}}$ with $\boldsymbol{C}_{\boldsymbol{\mu}(\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}})}$ we have that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$.

Proof. The proof of this theorem is motivated by the soundness proof in [2]. The idea of the proof is the following: Starting from two acyclic expressions $\boldsymbol{M_0} = \boldsymbol{M} \cong \boldsymbol{N} = \boldsymbol{N_0}$, we create expressions $\boldsymbol{M_1}, \dots, \boldsymbol{M_b}$ and $\boldsymbol{N_1}, \dots, \boldsymbol{N_{b'}}$ such that $\boldsymbol{M_{i+1}}$ is obtained from $\boldsymbol{M_i}$ via a replacement of encryption terms as described in condition (ii). Acyclicity ensures that the encrypting key of the replaced encryption terms will not occur anywhere else. Similarly for $\boldsymbol{N_{i+1}}$ and $\boldsymbol{N_i}$. We do this so that $\boldsymbol{M_b}$ and $\boldsymbol{N_{b'}}$ will differ only in key renaming. Then, by condition (ii), $[\![\boldsymbol{M_{i+1}}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M_i}]\!]_{\boldsymbol{\Phi}}$, and $[\![\boldsymbol{N_{i+1}}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N_i}]\!]_{\boldsymbol{\Phi}}$. But, $[\![\boldsymbol{M_b}]\!]_{\boldsymbol{\Phi}} = [\![\boldsymbol{N_{b'}}]\!]_{\boldsymbol{\Phi}}$ and therefore the theorem follows.

Now in more detail. Condition (ii) follows from (i) easily: for any set $\left\{\boldsymbol{C}_{\boldsymbol{\mu}(\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}})}\right\}_{\boldsymbol{i=1}}^{\boldsymbol{l}}$ provided by Proposition 9.3, the encrypting key of $\boldsymbol{C}_{\boldsymbol{\mu}(\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}})}$ is not contained in *S* hence it is not recoverable key of *M*. Therefore, while computing the pattern of *M'*, $\boldsymbol{C}_{\boldsymbol{\mu}(\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}})}$ will be replaced by the box $\square_{\boldsymbol{\mu}(\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}})}$, which is the same box as the one that replaces $\left\{\boldsymbol{N_{i_j}}\right\}_{\boldsymbol{K}}$ in *M* when the pattern of is computed. Hence $\boldsymbol{M} \cong \boldsymbol{M'}$ and therefore, since soundness is assumed, and *B-Keys*(*M'*) is not cyclic in *M'*, we have

$$[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}.$$

In order to prove that (i) follows from (ii), consider two equivalent valid expressions *M* and *N* such that $\boldsymbol{M} \cong \boldsymbol{N}$. Then, by definition, there exists a bijection $\boldsymbol{\sigma}$ on Keys(preserving $\equiv_{\boldsymbol{K}}$ such that *pattern*(*M*) = *pattern*(*N*$\boldsymbol{\sigma}$). This means that the "boxes" occurring in *pattern*(*M*) must

oc-cur in *pattern*($N\boldsymbol{\sigma}$) and vice-versa. Also, the subexpressions of *pattern*(*M*) and of *pattern*($N\boldsymbol{\sigma}$) outside the boxes must agree as well. Hence,

$$\boldsymbol{R-Keys(M) = R-Keys(N\sigma) = R-Keys(N)\sigma}.$$

Let $L_1, L_2, \ldots, L_b$ ($L_i \neq L_j$ if $i \neq j$) denote the keys in *B-Keys*(*M*), and let $L'_1, L'_2, \ldots, L'_{b'}$ ($L'_i \neq L'_j$ if $i \neq j$) denote the keys in *B-Keys*(*N*) $\boldsymbol{\sigma}$. *B-Keys*(*M*) and *B-Keys*(*N*) (and therefore *B-Keys*($N\boldsymbol{\sigma}$) as well) are not cyclic by hypothesis, so without loss of generality, we can assume that the L_i's and the L'_i's are numbered in such a way that L_i encrypts L_j (and L'_i encrypts L'_j) only if $i < j$ (for a more detailed argument about this, see [2]; intuitively this means that those keys in *B-Keys*(*M*) that are deeper in *M* have a higher number).

Consider now the set of expressions that are subexpressions of *M* or *N* and have the form $\{\boldsymbol{M'}\}_{\boldsymbol{L_i}}$ $\boldsymbol{or}$ $\{\boldsymbol{N'}\}_{\boldsymbol{L'_i}}$, and also, the set *S*. Condition (ii) then provides the set with elements of the form $\boldsymbol{C_{\mu(\{M'\}_{L_i})}}$ and $\boldsymbol{C_{\mu(\{N'\}_{L'_i})}}$.

Let $M_0 = M$. Let M_1 be the expression obtained from M_0 by replacing all subexpressions in M_0 of the form $\{\boldsymbol{M'}\}_{\boldsymbol{L_1}}$ by $\boldsymbol{C_{\mu(\{M'\}_{L_1})}}$ given by the assumption. Let then M_i, $i \geq 2$, be the expression obtained from $\boldsymbol{M_{i-1}}$ by replacing all subexpressions in $\boldsymbol{M_{i-1}}$ of the form $\{\boldsymbol{M'}\}_{\boldsymbol{L_i}}$ by $\boldsymbol{C_{\mu(\{M'\}_{L_i})}}$. We do this for all $i \leq b$ and it is easy to see that in M_b replacing the subexpressions of the form $\boldsymbol{C_{\mu(\{M'\}_{L_i})}}$ by $\square_{\boldsymbol{\mu(\{M'\}_{L_i})}}$ for all *i*, we arrive at *pattern*(*M*).

Note that in $\boldsymbol{M_{i-1}}$, L_i can only occur as an encrypting key. The reason for this is that if L_i is a subexpression of *M*, then it has to be encrypted with some non-recoverable key, otherwise L_i would be recoverable; moreover, it has to be encrypted with some key in *B-Keys*(*M*) because a subexpression of *M* is either recoverable or ends up in a box when we construct *pattern*(*M*). Now, the element in *B-Keys*(*M*) that encrypts L_i has to be an L_j with $j < i$. But, all subexpres-sions in *M* of the form $fM^0 g_{Lj}$ were already replaced by $\boldsymbol{C_{\mu(\{M'\}_{L_j})}}$ when we

constructed M_j. According to the properties listed in proposition 2.13, L_i may only appear in $\boldsymbol{C}_{\boldsymbol{\mu}(\{\boldsymbol{M}'\}_{\boldsymbol{L_j}})}$ as the encrypting key, and then $L_i = L_j$, a contradiction. So L_i cannot appear in $M_{i_j 1}$ in any other place than an encrypting key. Observe as well, that *R-Keys*(M_i) = *R-Keys*(M).

From assumption (ii), it follows then that $[\![\boldsymbol{M_{i-1}}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M_i}]\!]_{\boldsymbol{\Phi}}$, for all i, $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{b}$. Hence,

$$[\![\mathbf{M}]\!]_{\boldsymbol{\Phi}} = [\![\mathbf{M_0}]\!]_{\boldsymbol{\Phi}} \approx [\![\mathbf{M_b}]\!]_{\boldsymbol{\Phi}} \qquad (2.3)$$

Carrying out the same process for $N\boldsymbol{\sigma}$ through $(N\boldsymbol{\sigma})_0$, ($N\boldsymbol{\sigma}_1$, ..., $(N\boldsymbol{\sigma})_b{}'$ we arrive at

$$[\![\boldsymbol{\sigma}]\!]_{\boldsymbol{\Phi}} = [\![(\mathbf{N\sigma})_{\mathbf{0}}]\!]_{\boldsymbol{\Phi}} \approx [\![(\mathbf{N\sigma})_{\mathbf{b'}}]\!]_{\boldsymbol{\Phi}} \qquad (2.4)$$

Since we supposed that $M \cong N$, that is, *pattern*(M) = *pattern*($N\boldsymbol{\sigma}$), and therefore $M\ b$ » *pattern*(M) and $(N\boldsymbol{\sigma})_b 0$ = *pattern*($N\boldsymbol{\sigma}$), we have

$$[\![\mathbf{M_b}]\!]_{\boldsymbol{\Phi}} = [\![(\mathbf{N\sigma})_{\mathbf{b'}}]\!]_{\boldsymbol{\Phi}} \qquad (2.5)$$

Then, it is clearly true that

$$[\![\mathbf{N}]\!]_{\boldsymbol{\Phi}} \approx [\![\mathbf{N\sigma}]\!]_{\boldsymbol{\Phi}} \qquad (2.6)$$

because permuting the keys in N does not have any effect in the distributions. Putting together Equations (2.3), (2.4), (2.5) and (2.6) the soundness result follows:

$$[\![\mathbf{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\mathbf{N}]\!]_{\boldsymbol{\Phi}}$$

Remark 9.3. The reason there is no similar general theorem for key-cycles and KDM-like security is that this general soundness theorem tells us in which conditions the several steps of the Abadi-Rogaway hybrid argument can be carried out. One of the conditions is that by doing one step of replacement, we must obtain equivalent interpretations, provided that we have the appropriate security notion. However, in our theorem using KDM security to solve the key-cycles issue, there is only one step of replacement! All the replacements of undecryptable terms is done at once. Therefore, in a general theorem (without assuming a specific security level), the condition of the theorem would have to be exactly what we would want to prove.

Example 9.12 (Type-1 Soundness). The soundness theorem that was presented earlier for type-1 encryption schemes is a special case of the theorem above. In this case $\mathrm{Exp}_v = \mathrm{Exp}$; the equivalence relation $\equiv_{\boldsymbol{C}}$ is proper; and the equivalence relation $\equiv_{\boldsymbol{K}}$ is trivial here, all keys are equivalent. The elements $\boldsymbol{\mu} \in Q_{\mathrm{Enc}}$ are in one-to-one correspondence with the possible length, so the patterns that we obtain this way are essentially the same what we defined earlier, and the equivalence of expressions will be $\cong_{\mathbf{1}}$ that we also defined there. In order to see that condition (ii) of the general soundness theorem is satisfied for type-1, we will use the following equivalent definition of type-1 secure encryption schemes: we can also say that an encryption-scheme is *type-1 secure* if no PPT adversary A can distinguish the pair of oracles ($E(k, \bullet, \bullet, 0)$; $E(k', \bullet, \bullet, 0)$) and ($E(k, \bullet, \bullet, 1)$; $E(k, \bullet, \bullet, 1)$) as k and k' are independently generated, that is, for all PPT adversaries A:

$$\mathbf{Pr}\left[\boldsymbol{k}, \boldsymbol{k'} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\bullet,\bullet,\mathbf{0}),\boldsymbol{E}(\boldsymbol{k'},\bullet,\bullet,\mathbf{0})}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}\right] -$$

$$\mathbf{Pr}[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \boldsymbol{A}^{\boldsymbol{E}(\boldsymbol{k},\bullet,\bullet,\mathbf{1}),\boldsymbol{E}(\boldsymbol{k'},\bullet,\bullet,\mathbf{1})}(\mathbf{1}^{\boldsymbol{\eta}}) = \mathbf{1}] \leq \boldsymbol{neg}(\boldsymbol{\eta})$$

where the oracle $\boldsymbol{E}(\boldsymbol{k},\bullet,\bullet,\mathbf{0})$, upon the submission of two messages with equal lengths encrypts the first, and the oracle $\boldsymbol{E}(\boldsymbol{k},\bullet,\bullet,\mathbf{1})$ encrypts the second.

To show that condition (ii) of Theorem 9.14 holds, we first have to choose for $\{\boldsymbol{C_v}\}_{\boldsymbol{v}\in\boldsymbol{\mu}(\boldsymbol{C})}$ a given set $\boldsymbol{C} = \{\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}\}_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}}$. We can choose any family $\{\boldsymbol{C_v}\}_{\boldsymbol{v}\in\boldsymbol{\mu}(\boldsymbol{C})}$ such that all the $\boldsymbol{C_v}$ are encrypted with the same key, let's call it L_0, that is not present in any of the $\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}$ (neither in M). This is possible, because, as it is easy to check, $\boldsymbol{v_{key}} = \boldsymbol{Keys}\ \mathbf{for\ all}\ \boldsymbol{v} \in \boldsymbol{Q_{Enc}}$. Then, let M be as in condition (ii). We need to show that if $\{\{\boldsymbol{N_{i_j}}\}_{\boldsymbol{L}}\}_{\boldsymbol{j}=\mathbf{1}}^{\boldsymbol{l}} \subseteq \boldsymbol{C}$ and if we denote by M' the expression obtained from M by replacing each $\{\boldsymbol{N_{i_j}}\}_{\boldsymbol{L}}$ with $\boldsymbol{C}_{\boldsymbol{\mu}(\{\boldsymbol{N_{i_j}}\}_{\boldsymbol{L}})}$, then $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$.

Suppose that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \not\approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$, which means that there is an adversary A that is able to distin-guish the two distributions, that is

$$\mathbf{Pr}\left[\boldsymbol{x} \leftarrow [\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}_{\boldsymbol{\eta}}} : \boldsymbol{A}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{x}) = \mathbf{1}\right] - \mathbf{Pr}[\boldsymbol{x} \leftarrow [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}_{\boldsymbol{\eta}}} : \boldsymbol{A}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{x}) = \mathbf{1}]$$

is a non-negligible function of η. Let us show that this contradicts type-1 security. To this end, let us construct an adversary that can distinguish between the two pair of oracles above. This adversary is the following probabilistic algorithm that access to the oracles *f* and *g*:

algorithm B $^{f,g}(\mathbf{1}^{\eta}; M)$

for $K \in Keys(M) \setminus \{L, L_0\}$ do $\boldsymbol{\tau}(K) \leftarrow K(\mathbf{1}^{\eta})$

$\boldsymbol{y} \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M})$

$\boldsymbol{b} \leftarrow (\mathbf{1}^{\eta}, \boldsymbol{y})$

return b

algorithm CONVERT2(*N*)

if $N = K$ where $K \in$ Keys then return $\boldsymbol{\tau}(K)$

if $N = B$ where $B \in$ Blocks then return B

if $N = (M_1; M_2)$ then

$\boldsymbol{x} \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M_1})$

$y \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M_2})$

return $[x; y]$

if $N = \{\boldsymbol{M_1}\}_{\boldsymbol{L}}$ then

$\boldsymbol{x} \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M_1})$

$y \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M_v})$ (where $\boldsymbol{C_{\mu(\{M_1\}_L)}} = \{\boldsymbol{M_v}\}_{\boldsymbol{L_0}}$)

$\boldsymbol{z} \leftarrow \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$

return *z*

if $N = \{\boldsymbol{M_1}\}_{\boldsymbol{L_0}}$ then

$\boldsymbol{x} \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M_1})$

$\boldsymbol{y} \leftarrow \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{x})$

return *y*

if $N = \{\boldsymbol{M_1}\}_{\boldsymbol{K}}$ $(\boldsymbol{K} \notin \{\boldsymbol{L}, \boldsymbol{L_0}\})$ then

$\boldsymbol{x} \leftarrow \boldsymbol{CONVERT2}(\boldsymbol{M_1})$

$\boldsymbol{y} \leftarrow \boldsymbol{E}(\boldsymbol{\tau}(\boldsymbol{K}), \boldsymbol{x})$

return *y*

Note that the algorithm CONVERT2 does almost the same as the algorithm CONVERT in Figure 9.1, except that while CONVERT carries out all the necessary encryptions, CONVERT2 makes the oracles carry out the encryptions for *L* and L_0. Therefore, in the case,

when the pair of oracles (*f; g*) is (*E*(*k*, •, •, 0)*; E*(*k*′, •, •, 0)), then CONVERT2(*M*) will be a random sample from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}_\eta}$, whereas if the pair of oracles used is (*E*(*k*, •, •, 1),*E*(*k*, •, •, 0)), then CONVERT2(*M*) will be a random sample from $[\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}_\eta}$. Thus,

$$\mathbf{Pr}\left[k, k' \leftarrow K(1^\eta) : B^{E(k,\bullet,\bullet,0),E(k',\bullet,\bullet,0)}(1^\eta, M) = 1\right] - \mathbf{Pr}[x \leftarrow [\![M]\!]_{\Phi_\eta} : A(1^\eta, x) = 1]$$

and

$$\mathbf{Pr}\left[k \leftarrow K(1^\eta) : B^{E(k,\bullet,\bullet,1),E(k,\bullet,\bullet,1)}(1^\eta, M) = 1\right] - \mathbf{Pr}[x \leftarrow [\![M']\!]_{\Phi_\eta} : A(1^\eta, x) = 1]$$

But, according to our assumption, $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ and $[\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$ can be distinguished, that is,

$$\mathbf{Pr}\left[x \leftarrow [\![M]\!]_{\Phi_\eta} : A(1^\eta, x) = 1\right] - \mathbf{Pr}[x \leftarrow [\![M']\!]_{\Phi_\eta} A(1^\eta, x) = 1]$$

is a non-negligible function of $\boldsymbol{\eta}$ and so, there is an adversary $B^{f,g}(1^\eta, x)$ such that

$$\mathbf{Pr}\left[k, k' \leftarrow K(1^\eta) : B^{E(k,\bullet,\bullet,0),E(k',\bullet,\bullet,0)}(1^\eta, M) = 1\right] - \mathbf{Pr}[k \leftarrow K(1^\eta) : B^{E(k,\bullet,\bullet,1),E(k,\bullet,\bullet,1)}(1^\eta, M) = 1]$$

is also a non-negligible function of $\boldsymbol{\eta}$. This implies that our scheme cannot be type-1 secure, which contradicts the assumption. Hence, we cannot have $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$. Hence, condition (ii) of the general soundness theorem is satisfied, so soundness holds for the type-1 case.

Example 9.13 (Type-2 Soundness). The soundness theorem that was presented earlier for type-2 encryption schemes is also a special case of the theorem above. In this case Exp_V = Exp; the equivalence relation $\equiv_C$ is is proper; and the equivalence relation $\equiv_K$ is trivial here, all keys are equivalent. The elements $\boldsymbol{\mu} \in Q_{\mathrm{Enc}}$ are in one-to-one correspondence with the keys, so we can say $Q_{\mathrm{Enc}} \equiv$ Keys, and thus the boxes are labelled with keys. In this case Φ gives an interpretation in the computational setting. Then for a set $\boldsymbol{C} = \{\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}$ as in condition (ii) of the theorem, we can take $\boldsymbol{C_{L_i}} := \{\boldsymbol{0}\}_{\boldsymbol{L_i}}$, and then condition (ii) is satisfied, because the following proposition holds:

Proposition 9.4. *Consider an expression M, and a key L* ∈ *Keys*(*M*).

Suppose that for some expressions M_1, M_2, ..., $M_l \in Exp$, $\{\boldsymbol{M_1}\}_L, \{\boldsymbol{M_2}\}_L, \ldots, \{\boldsymbol{M_l}\}_L \boldsymbol{M}$ *, and assume also that L does not occur anywhere else in M. Then, denoting by M' the expression that we get from M by replacing each of* $\{\boldsymbol{M_i}\}_L$ *that are not contained in any of* $\boldsymbol{M_j}(\boldsymbol{j \neq i})$ *by* $\{\boldsymbol{0}\}_L$, $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$ *holds when the expressions are interpreted with a type-2 encryption scheme.*

Proof. We can assume, without loss of generality, that $\{\boldsymbol{M_i}\}_L$ is a subexpression of $\{\boldsymbol{M_j}\}_L$ only if $i < j$. Suppose that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \not\approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$, which means that there is an adversary A that distinguishes the two distributions, that is

$$\mathbf{Pr}\left(\boldsymbol{x} \leftarrow [\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi_\eta}} : \boldsymbol{A}(\boldsymbol{1^\eta}, \boldsymbol{x}) = \boldsymbol{1}\right) - \mathbf{Pr}(\boldsymbol{x} \leftarrow [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi_\eta}} : \boldsymbol{A}(\boldsymbol{1^\eta}, \boldsymbol{x}) = \boldsymbol{1})$$

is a non-negligible function of $\boldsymbol{\eta}$. Let us show that this contradicts type-2 security. To this end, let us construct an adversary that can distinguish between the oracles $E(k; \bullet)$ and $E(k; 0)$. This adversary is the following probabilistic algorithm that access to the oracle f:

algorithm $B^f(\boldsymbol{1^\eta}; M)$
for $K \in$ *2 Keys*$(M) \setminus \{L\}$ do $\boldsymbol{\tau}(\boldsymbol{K}) \leftarrow K(\boldsymbol{1^\eta})$
$\boldsymbol{y} \leftarrow$ CONVERT2(M)
$\boldsymbol{b} \leftarrow A(\boldsymbol{1^\eta}; y)$ return b
algorithm CONVERT2(N)
if $N = K$ where $K \in$ Keys then return $\boldsymbol{\tau}(K)$
if $N = B$ where $B \in$ Blocks then
return B
if $N = (N_1; N_2)$ then
$\boldsymbol{x} \leftarrow$CONVERT2$(N_1)$ $\boldsymbol{y} \leftarrow$CONVERT2$(N_2)$ return $[x; y]$
if $N = \{\boldsymbol{N_1}\}_L$ then
$\boldsymbol{x} \leftarrow$ CONVERT2(N_1) y $\boldsymbol{y} \leftarrow f(x)$
return y
if $N = \{\boldsymbol{N_1}\}_K$ $(K \neq L)$ then
$\boldsymbol{x} \leftarrow$ CONVERT2(N_1)
$\boldsymbol{y} \leftarrow E(\boldsymbol{\tau}(K); x)$ return y

Note that the algorithm CONVERT2 does almost the same as the algorithm CONVERT in Figure 9.1, except that while CONVERT carries out all necessary encryptions, CONVERT2 makes the oracles carry out the encryptions for *L*. Therefore, in the case, when the oracle *f* is *E*(*k;* •), then CONVERT2(*M*) will be a random sample from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi_\eta}}$, whereas if the oracle used is *E*(*k;* 0), then CONVERT2(*M*) will be a random sample from $[\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi_\eta}}$. Thus,

$$\mathbf{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \ \boldsymbol{B}^{\boldsymbol{E}(\boldsymbol{k};\,\bullet)}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{M}) = \mathbf{1}\right] = \mathbf{Pr}\left[\boldsymbol{x} \leftarrow [\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi_\eta}} : \boldsymbol{A}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{x}) = \mathbf{1}\right]$$

And

$$\mathbf{Pr}\left[\boldsymbol{k} \leftarrow \boldsymbol{K}(\mathbf{1}^{\boldsymbol{\eta}}) : \ \boldsymbol{B}^{\boldsymbol{E}(\boldsymbol{k};\,\mathbf{0})}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{M}) = \mathbf{1}\right] = \mathbf{Pr}\left[\boldsymbol{x} \leftarrow [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi_\eta}} : \boldsymbol{A}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{x}) = \mathbf{1}\right]$$

But, according to our assumption, $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ and $[\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$ can be distinguished, that is,

$$\mathbf{Pr}\left[\boldsymbol{x} \leftarrow [\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi_\eta}} : \ \boldsymbol{A}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{M}) = \mathbf{1}\right] - \mathbf{Pr}\left[\boldsymbol{x} \leftarrow [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi_\eta}} : \boldsymbol{A}(\mathbf{1}^{\boldsymbol{\eta}}, \boldsymbol{x}) = \mathbf{1}\right]$$

is a non-negligible function of $\boldsymbol{\eta}$ and so, there is an adversary $B^{f}(\mathbf{1}^{\boldsymbol{\eta}}, \bullet)$ that can distinguish the oracles *E*(*k*, •) and *E*(*k*, 0), for randomly generated keys *k*. This implies that our scheme cannot be type-2 secure, which contradicts the assumption. Hence, we cannot have $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{M'}]\!]_{\boldsymbol{\Phi}}$.

Hence, condition (ii) of the general soundness theorem is satisfied, so soundness holds for the type-2 case.

Example 9.14 (Soundness for One-Time Pad). In order to see that the formal treatment of Section sec:OTP is a special case of the general formalism, take $\equiv_{\boldsymbol{C}}$ so that two encryption terms are equivalent, iff (again) the encryption terms have the same encrypting key. The equivalence of keys, $\equiv_{\boldsymbol{K}}$ is defined with the help of a length-function *l* on the keys: two keys are equivalent iff they have the same length. The boxes will again be indexed by the encrypting keys. Then for a set $\boldsymbol{C} = \{\{\boldsymbol{N_i}\}_{\boldsymbol{L_i}}\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}$ as in condition (ii), take $\boldsymbol{C_{L_i}} \coloneqq \{\mathbf{0}_{\boldsymbol{l(L_i)-3}}\}_{\boldsymbol{L_i}}$ (where $\mathbf{0}_{\boldsymbol{l(L_i)-3}}$ means *l*(*L*$_i$) - 3 many 0's). It is not hard to check that within

this setting, condition (ii) of the soundness theorem is satisfied, which is an immediate consequence of the following proposition:

Proposition 9.5. *Consider a valid expression* $M \in Exp_{\mathrm{OTP}}$*, and a key* $K_0 \in Keys(M)$*. Sup-pose that for some expression* M_0*,* $\{\boldsymbol{M_0}\}_{\boldsymbol{K_0}}$ *is a subexpression of M, and assume also that* K_0 *does not occur anywhere else in M. Then, denoting by M' the expression that we get from M by replacing* $\{\boldsymbol{M_0}\}_{\boldsymbol{K_0}}$ *with* $\{\mathbf{0}_{\boldsymbol{l(L_i)-3}}\}_{\boldsymbol{K_0}}$ *(where* $\mathbf{0}_{\boldsymbol{l(L_i)-3}}$ *denotes as string consisting of* $l(K_0)$ - 3 *many* 0*'s), the following is true when* Φ *is the interpretation for OTP:*

$$[\![\mathbf{M}]\!]_{\boldsymbol{\Phi}} = [\![\mathbf{M'}]\!]_{\boldsymbol{\Phi}} \qquad (2.7)$$

Proof. The basic properties of the OTP ensure that $\boldsymbol{\Phi}(\{\boldsymbol{M_0}\}_{\boldsymbol{K_0}})$ is evenly distributed over the set of $l(K_0)$ long strings ending with 110, no matter what M_0 is. So the distribution of $\boldsymbol{\Phi}(\{\boldsymbol{M_0}\}_{\boldsymbol{K_0}})$ agrees with the distribution of $\boldsymbol{\Phi}(\{\mathbf{0}_{\boldsymbol{l(K_0)-3}}\}_{\boldsymbol{K_0}})$. Also, since K_0 is assumed not to occur anywhere else, $\boldsymbol{\Phi_M}(K_0)$ is independent of the interpretation of the rest of the expression *M*, and therefore, $\boldsymbol{\Phi}(\{\boldsymbol{M_0}\}_{\boldsymbol{K_0}})$ and $\boldsymbol{\Phi}(\{\mathbf{0}_{\boldsymbol{l(K_0)-3}}\}_{\boldsymbol{K_0}})$ are both independent of the interpretation of the rest of the expression. Hence, replacing $\boldsymbol{\Phi}(\{\boldsymbol{M_0}\}_{\boldsymbol{K_0}})$ with $\boldsymbol{\Phi}(\{\mathbf{0}_{\boldsymbol{l(K_0)-3}}\}_{\boldsymbol{K_0}})$ will not effect the distribution.

Parsing Process

Let us present the the chapter will be useful in the course of proving the completeness results. The idea can be summarised as follows: Given a sample element $\boldsymbol{x} \leftarrow [\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$, *x* is built from blocks and randomly generated keys which are paired and encrypted. Some of the keys that were used for encryption when *x* was built might be explicitly contained in *x*, and in this case, using these keys, we can decrypt those ciphers that were encrypted with these revealed keys. The problem is though, that looking at *x*, it might not be possible to tell where blocks, keys, ciphers and pairs are in the string of bits, since we did not assume in general that we tag strings as we did for OTP. However, and we will exploit this fact repeatedly in our proofs, if we know that *x* was

sampled from $[\![M]\!]_{\Phi}$ for a fixed, known expression *M*, then by looking at *M*, we can find in *x* the locations of blocks, keys, ciphers and pairs, and we can also tell from *M*, where the key decrypting a certain cipher is located. Later we will present a machinery that, using the form of an expression *M*, extracts from an $x \leftarrow [\![M]\!]_{\Phi}$ everything that is possible via decryption and depairing, and distributes the extracted elements over a special Cartesian product of copies of $\overline{strings}$.

Throughout this section, we assume that $\Delta = (Exp_V, \equiv_K, \equiv_C)$ and an interpretation Φ in a general symmetric encryption scheme $\Pi = (\{K_i\}_{i \in I}, E, D, \approx)$ is given.

In this chapter we will use the notion of *subexpression occurrence* of/in *M*. This means a subexpression together with its position in *M*. The reason for this distinction is that a subexpression can occur several times in *M*, and we want to distinguish these occurrences. But, to avoid cumbersome notation, we will denote the subexpression occurrence just as the subexpression itself. Let us start by defining the notion of 0-level subexpression occurrences of an expression *M*:

Definition 9.39 (Level 0 Subexpression Occurrences). For an expression *M*, let us call *level 0 subexpression occurrences* all those subexpression occurrences in *M* that are not encrypted. Let $sub_0(M)$ denote the set of all level 0 subexpression occurrences in *M*. let us write $N \sqsubseteq_0 M$ if *N* is a level 0 subexpression occurrence of *N* in *M*.

For an element $x \leftarrow [\![M]\!]_{\Phi}$, the first thing to do is to extract everything that is not encrypted, which means that we have to break up all pairs in *x*, and replace them with mathematical pairs. This process reveals the unencrypted blocks, keys and ciphers in *x* (i.e., the computational or statistical realisations of the 0-level subexpression occurrences).

Definition 9.40 (Blowup Function). For each valid expression *M*, we define the *blowup funcion* $B(M)$, on $\overline{strings}$ inductively as follows:

$$B(K)x := x \quad \text{for } K \text{ key } B(B)x := x \quad \text{for } B \text{ block}$$

$$B((M_1; M_2))x := (B(M_1) \oplus B(M_2)) \circ [\bullet\,,\bullet]^{-1}(x)\; B(\{N\}_K)x := x:$$

Where $B(M_1) \oplus B(M_2)$ denotes the function $(x; y) \mapsto (B(M_1)x, B(M_2)y)$.

The element $B(M)x$ is an element of $\boldsymbol{\tau}_0(M)$, which we define inductively the following way:

Definition 9.41 (Associated 0-Tree). The *0-tree associated* to a pair of expressions N and M whenever $N \nu_0 M$, will be denoted by $\boldsymbol{\tau}_0(N; M)$, and we define it inductively as follows:

$$\boldsymbol{\tau}_0(K; M) := \overline{\boldsymbol{strings}}\ \boldsymbol{\tau}_0(B; M) := \overline{\boldsymbol{strings}}$$

$$\boldsymbol{\tau}_0((M_1; M_2); M) := \boldsymbol{\tau}_0(M_1; M) \times \boldsymbol{\tau}_0(M_2; M)$$

$$\boldsymbol{\tau}_0(\{\boldsymbol{M'}\}_K, M) := \overline{\boldsymbol{strings}}$$

Let $\boldsymbol{\tau}_0(M) := \boldsymbol{\tau}_0(M; M)$.

Remember, that we do not identify $(\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \times \overline{\boldsymbol{strings}}$ with $\overline{\boldsymbol{strings}} \times (\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}})$.

Note also that for expressions $N \nu_0 M'$ and $N \nu_0 M$, we have that $\boldsymbol{\tau}_0(N; M^{\,0}) = \boldsymbol{\tau}_0(N; M)$. Nevertheless, we included M in the definition of $\boldsymbol{\tau}_0$ since for higher order trees, which will be defined later, the M in the second argument will make a difference.

Example 9.15. For the expression

$$\boldsymbol{M} = \left(\left(\{\boldsymbol{0}\}_{K_6}, \left\{\{\boldsymbol{K_7}\}_{K_1}\right\}_{K_4}\right), \left(\left(\boldsymbol{K_2}, \left\{\left(\{\boldsymbol{001}\}_{K_3}, \{\boldsymbol{K_6}\}_{K_5}\right)\right\}_{K_5}\right), \{\boldsymbol{K_5}\}_{K_2}\right)\right),$$

$$\boldsymbol{sub_0(M)} = \left\{\begin{array}{l}\left(\{\boldsymbol{0}\}_{K_6}, \left\{\{\boldsymbol{K_7}\}_{K_1}\right\}_{K_4}\right), \boldsymbol{K_2}, \left\{\{\boldsymbol{001}\}_{K_3}, \{\boldsymbol{K_6}\}_{K_5}\right\}_{K_5}, \{\boldsymbol{K_5}\}_{K_2}, \left(\{\boldsymbol{0}\}_{K_6}, \left\{\{\boldsymbol{K_7}\}_{K_1}\right\}_{K_4}\right), \\ \left(\boldsymbol{K_2}, \left\{\left(\{\boldsymbol{001}\}_{K_3}, \{\boldsymbol{K_6}\}_{K_5}\right)\right\}_{K_5}\right), \left(\left(\boldsymbol{K_2}, \left\{\left(\{\boldsymbol{001}\}_{K_3}, \{\boldsymbol{K_6}\}_{K_5}\right)\right\}_{K_5}\right), \{\boldsymbol{K_5}\}_{K_2}\right), \boldsymbol{M}\end{array}\right.$$

and

$$\boldsymbol{\tau_0(M)} = (\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \times ((\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \times \overline{\boldsymbol{strings}}).$$

Blocks, keys and ciphers are replaced by $\overline{\boldsymbol{strings}}$, pairs are replaced by $\times$. An element x sampled from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ looks like

$$[\,[\boldsymbol{c_1}, \boldsymbol{c_2}], [\,[\boldsymbol{k}, \boldsymbol{c_3}], \boldsymbol{c_4}]\,]$$

where $\boldsymbol{c_1}$ is a sample from $[\![\{\mathbf{0}\}_{K_6}]\!]_{\Phi}$, $\boldsymbol{c_2}$ is a sample from $[\![\{\{K_7\}_{K_1}\}_{K_4}]\!]_{\Phi}$, k is a sample from $[\![K_2]\!]_{\Phi}$, c_3 is a sample from $[\![(\{\{001\}_{K_3},\{K_6\}_{K_5})\}_{K_5}]\!]_{\Phi}$, and c_4 is a sample from $[\![\{K_5\}_{K_2}]\!]_{\Phi}$. When we apply the blow-up function to this element x, we obtain

$$((c_1, c_2), (k, c_3), c_4))$$

which is an element of $\tau_0(M)$.

Proposition 9.6. *For an expression M, if* $x \leftarrow [\![M]\!]_{\Phi}$, *then* $B(M)(x) \in \tau_0(M)$.

Proof. Immediate from the definitions of B and τ_0. Perhaps it is even clearer if we label the copies of $\overline{strings}$ in $T_0(M)$ with the formal expres-sions that they belong to:

$$\tau_0'(K, M) := \overline{strings}_K$$
$$\tau_0'(B, M) := \overline{strings}_B$$
$$\tau_0'((M_1, M_2)M) := \tau_0'(M_1, M) \times \tau_0'(M_2, M)$$
$$\tau_0'(\{M'\}_K, M) := \overline{strings}_{\{M'\}_K}$$

In our example,

$$\tau_0'(M, M) =$$
$$\left(s_{\{0\}_{K_6}} \times s_{\{\{K_7\}_{K_1}\}_{K_4}}\right) \times ((s_{K_2} \times s_{\{(\{001\}_{K_3},\{K_6\}_{K_5})\}_{K_5}}) \times s_{\{K_5\}_{K_2}}),$$

where we used s as a shorthand for $\overline{strings}$.

In the previous example, c_4 is a random sample from $[\![M\{K_5\}_{K_2}]\!]_{\Phi}$, and the function that projects onto the last copy of $\overline{strings}$. in $\tau_0(M)$, namely, onto $\overline{strings}_{\{K_5\}_{K_2}}$, extracts c_4 from the blow-up. Similarly, projecting onto the other copies of $\overline{strings}$, we extract samples form $[\![\{\mathbf{0}\}_{K_6}]\!]_{\Phi}$, $[\![\{\{K_7\}_{K_1}\}_{K_4}]\!]_{\Phi}$etc. To implement this idea in the general situation, we define what we can call the "0-Get Function" $G_0(N; M)$ for an expression M and a subexpression occurrence N, whenever N is not encrypted in M. For $x \leftarrow [\![M]\!]_{\Phi}$, the purpose of $G_0(N; M)$ is to extract from $B(M)x$ the sample of $[\![N]\!]_{\Phi}$ that was used for computing x. The precise definition is the following:

Definition 2.42 (0-Get Function). For subexpression occurrences $N \sqsubseteq_0 N' \sqsubseteq_0 M$, we define the *0-get function associated* to the triple (N, N', M), $\mathcal{G}_0(N, N', M) : \mathcal{T}_0(N', M) \to \mathcal{T}_0(N, M)$ inductively in N' as follows:

$$\mathcal{G}_0(N, N, M) := \mathrm{id}_{\mathcal{T}_0(N,M)}$$

$$\mathcal{G}_0(N, (M_1, M_2), M) := \begin{cases} \mathcal{G}_0(N, M_1, M) \circ \pi^1_{\mathcal{T}_0(M_1,M)\times\mathcal{T}_0(M_2,M)} & \text{if } N \text{ occurs in } M_1, \\ \mathcal{G}_0(N, M_2, M) \circ \pi^2_{\mathcal{T}_0(M_1,M)\times\mathcal{T}_0(M_2,M)} & \text{otherwise} \end{cases}$$

We define $\mathcal{G}_0(N, M) := \mathcal{G}_0(N, M, M)$.

Example 9.16. In the example,

$$\boldsymbol{G_0\big(\{0\}_{K_6}, M\big), G_0\left(\left\{\{K_7\}_{K_1}\right\}_{K_4}, M\right)}\boldsymbol{: \tau_0(M) \to \overline{strings}}$$

$$\boldsymbol{G_0\big(\{0\}_{K_6}, M\big)((x_1, x_2), \big((x_3, x_4), x_5\big) = x_1,}$$

$$\boldsymbol{G_0\left(\left\{\{K_7\}_{K_1}\right\}_{K_4}, M\right)\big(\,(x_1, x_2), \big((x_3, x_4), x_5\big) = x_2,}$$

etc; that is, $\boldsymbol{G_0\big(\{0\}_{K_6}, M\big)}$ does the projection onto $\boldsymbol{\overline{strings}_{\{0\}_{K_6}}}$, $\boldsymbol{G_0\left(\left\{\{K_7\}_{K_1}\right\}_{K_4}, M\right)}$ does the projection onto $\boldsymbol{\overline{strings}_{\left\{\{K_7\}_{K_1}\right\}_{K_4}}}$, etc.

Observe, that for two expressions M and N, if $\boldsymbol{\tau_0(M) = \tau_0(N)}$, then for any M' ∈ $\boldsymbol{sub_0(M)}$, there is a ynique N' ∈ $\boldsymbol{sub_0(N)}$, such that $\boldsymbol{G_0(M', M) = G_0(N', N)}$. This motivates the following definition:

Definition 9.43 (Same Position of Subexpression Occurrences). For two expressions *M* and *N*, if $\boldsymbol{\tau_0(M)} = \boldsymbol{\tau_0(N)}$, we say that $M' \in sub_0(M)$ and $N' \in sub_0(M)$ *are in the same position at level 0*, if $G_0(M', M) = G_0(N', N)$*:*

Let $\Gamma_0(N; M) : sub_0(M) \to sub_0(N)$ denote the unique bijection such that $G_0(M', M) = G_0(\Gamma_0 (N; M)M', N)$ for all $M' \in sub_0(M)$.

Example 9.17. Let $N = ((0, 0), ((0, 0), 0)$. Then, if *M* denotes the expression from the previous examples, $\boldsymbol{\tau_0(N) = \tau_0(M)}$. Enumerating the 0's in *N*, we get the subexpression occurrences $0_1 = 0$, $0_2 = 0$, $0_3 = 0$, $0_4 = 0$ and $0_5 = 0$, with $N = ((0_1 , 0_2), ((0_3, 0_4), 0_5)$. We have that:

$$\boldsymbol{\Gamma_0(N, M)\{0\}_{K_6} = 0_1}$$

$$\boldsymbol{\Gamma_0(N, M)\left\{\{K_7\}_{K_1}\right\}_{K_4} = 0_2}$$

$$\boldsymbol{\Gamma_0(, M)K_2 = 0_3}$$

$$\boldsymbol{\Gamma_0(N, M)\big\{(\{001\}_{K_3}, \{K_6\}_{K_5})\big\}_{K_5} = 0_4}$$

$$\Gamma_0(\boldsymbol{N}, \boldsymbol{M})\{\boldsymbol{K_5}\}_{\boldsymbol{K_2}} = \boldsymbol{0_5}$$
$$\Gamma_0(\boldsymbol{N}, \boldsymbol{M})(\{\boldsymbol{0}\}_{\boldsymbol{K_6}}, \left(\{\{\boldsymbol{K_7}\}_{\boldsymbol{K_1}}\}_{\boldsymbol{K_4}}\right) = (\boldsymbol{0_1}, \boldsymbol{0_2})$$

etc.

For an expression *M*, let C_M denote the set of all those subexpression occurrences in *M* which are ciphers encrypted by recoverable keys, i.e.,

$$\boldsymbol{C_M} = \{\{\boldsymbol{M'}\}_{\boldsymbol{K}} \sqsubseteq \boldsymbol{M} \mid \{\boldsymbol{M'}\}_{\boldsymbol{K}} \in \boldsymbol{vis}(\boldsymbol{M}) \in \boldsymbol{R} - \boldsymbol{Keys}(\boldsymbol{M})\}$$

We emphasise that in the previous definition we are referring to subexpression occurrences, that is, if an encryption term is encrypted with a recoverable key occurs twice in *M*, then it will be listed twice in C_M . Since we assume that the elements of this set are encrypted by recoverable keys, it is possible to decrypt these elements one after the other, using only information contain-ing *M*. Therefore, it is possible to enumerate the elements of this set in an order in which we can decrypt them by taking keys from *M*, decrypting what is possible with these keys and hence revealing more keys and then decrypting again with those keys etc. Let the total number of this set be denoted by $c(M)$. Then

$$\boldsymbol{C_M} = \{\boldsymbol{C^1}, \boldsymbol{C^2}, \dots, \boldsymbol{C^{c(M)}}\}.$$

Note that this enumeration is not unique. Also, note that the numbering does not mean that you can decrypt the ciphers only in this order. Let $\boldsymbol{C^i_{key}}$ denote the key that is used in the encryption $\boldsymbol{C^i}$ and let $\boldsymbol{C^i_{key}}$ denote the encrypted expression.

Example 9.18. In our example, the only possible way to enumerate is

$$\boldsymbol{C^1} = \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}$$
$$\boldsymbol{C^2} = \{(\{\boldsymbol{001}\}_{\boldsymbol{K_3}}, \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}}\}_{\boldsymbol{K_5}}$$
$$\boldsymbol{C^3} = \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}}$$
$$\boldsymbol{C^4} = \{\boldsymbol{0}\}_{\boldsymbol{K_6}}.$$

Now, to each expression *M*, we can associate the "1-Decrypting Function" $D_1(M)$. It acts on $\boldsymbol{\tau_0}(M)$ and works as follows: for any $t \in$

$\boldsymbol{\tau_0}(M)$, the function $D_1(M)$ extracts $G_0(C^1, M)t$ from $\overline{\boldsymbol{strings}}_{\boldsymbol{C^1}}$, $G0(\boldsymbol{C^1_{key}}, M)t$ from $\overline{\boldsymbol{strings}}_{\boldsymbol{C^1_{key}}}$, and with the latter decrypts the former if that is possible (namely, if they are of the right form: the former a cipher and the latter a key). The result is then broken into mathematical pairs, and what we get this way is put in the last component of the set $\overline{\boldsymbol{strings}} \times \{\boldsymbol{0}\} \times \boldsymbol{\tau_0}(\boldsymbol{C^1_{text}})$, while $G0(\boldsymbol{C^1_{key}}, M)t$ goes into the first component. That is, the following element is created:

$$(\boldsymbol{G_0}\left(\boldsymbol{C^1_{key}}, \boldsymbol{M}\right)\boldsymbol{t}, \boldsymbol{0}, \boldsymbol{B}\left(\boldsymbol{C^i_{text}}\right)\left(\boldsymbol{D}\left(\boldsymbol{G_0}\left(\boldsymbol{C^1_{key}}\right)\boldsymbol{t}, \boldsymbol{G_0}\left(\boldsymbol{C^1}, \boldsymbol{M}\right)\boldsymbol{t}\right)\right)).$$

If $(\boldsymbol{G_0}\left(\boldsymbol{C^1_{key}}, \boldsymbol{M}\right)\boldsymbol{t}, \boldsymbol{G_0}\left(\boldsymbol{C^1_{key}}\right)\boldsymbol{t}) \notin \boldsymbol{Dom_D}$, then $\boldsymbol{D_1}(\boldsymbol{M})$ outputs (0, 0, 0). The rest og $\boldsymbol{\tau_0}(\boldsymbol{M})$ is left untouched. Let us warn that for the similiarity of notations between the algorithm of the encryption scheme $D(\bullet\ ,\ \bullet)$, and the 1-Decrypting function $D_1(\bullet)$. This notation is convenient as $D_i(M)$ is the function that decrypts the ciphers encrypted with recoverable keys at level-*i*. We will always index this functions with the respective index *i* to avoid confusions.

Let us introduce the notation

$$\boldsymbol{\tau_0^{C^1}}(\boldsymbol{M}) = \{\boldsymbol{t} \in \boldsymbol{\tau_0}(\boldsymbol{M}) \mid (\boldsymbol{G_0}\left(\boldsymbol{C^1_{key}}, \boldsymbol{M}\right)\boldsymbol{t}, \boldsymbol{G_0}(\boldsymbol{C^1}, \boldsymbol{M})\boldsymbol{t}) \in \boldsymbol{Dom_D}\}.$$

Definition 9.44 (1-Decrypting Function). For expressions $N \sqsubseteq_{\boldsymbol{0}} M$, let us define the function $D_1(N; M)$ on $\boldsymbol{\tau_0}(M)$ inductively as follows: Let $t \in \boldsymbol{\tau_0}(M)$. Then

$$\boldsymbol{D_1}(\boldsymbol{K}, \boldsymbol{M})\boldsymbol{t} := \boldsymbol{G_0}(\boldsymbol{K}, \boldsymbol{M})\boldsymbol{t}$$
$$\boldsymbol{D_1}(\boldsymbol{B}, \boldsymbol{M})\boldsymbol{t} := \boldsymbol{G_0}(\boldsymbol{B}, \boldsymbol{M})\boldsymbol{t}$$
$$\boldsymbol{D_1}(\{\boldsymbol{M'}\}_{\boldsymbol{K}}, \boldsymbol{M})\boldsymbol{t} := \boldsymbol{G_0}(\{\boldsymbol{M'}\}_{\boldsymbol{K}}, \boldsymbol{M})\boldsymbol{t}\ \boldsymbol{if}\ \boldsymbol{K} \notin \boldsymbol{R - Keys}(\boldsymbol{M})$$
$$\boldsymbol{D_1}\left((\boldsymbol{M_1}, \boldsymbol{M_2}), \boldsymbol{M}\right)\boldsymbol{t} := (\boldsymbol{D_1}(\boldsymbol{M_1}, \boldsymbol{M})\boldsymbol{t}, \boldsymbol{D_1}(\boldsymbol{M_2}, \boldsymbol{M})\boldsymbol{t})$$

$$\mathcal{D}_1(C^j, M)t := \\ := \begin{cases} \left(\mathcal{G}_0(C^1_{\text{key}}, M)t, 0, \mathcal{B}(C^1_{\text{text}})\left(\mathcal{D}(\mathcal{G}_0(C^1_{\text{key}}, M)t, \mathcal{G}_0(C^1, M)t)\right)\right) & \text{if } t \in \mathcal{T}_0^{C^1}(M) \text{ and } j = 1 \\ (0,0,0) \qquad \text{if } t \notin \mathcal{T}_0^{C^1}(M) \text{ and } j = 1 \\ \mathcal{G}_0(C^j, M)t \qquad \text{if } j > 1 \end{cases}$$

Let us introduce the notation $D_1(M) := D_1(M, M)$. We remark, that it is not important how we define $\boldsymbol{D_1}(\boldsymbol{C^1}, \boldsymbol{M})\boldsymbol{t}$ when t $\notin \boldsymbol{\tau_0^{C^1}}(\boldsymbol{M})$, we will not need that. We chose (0,0,0) just for convenience.

Example 9.19. In our running example we have

$$\boldsymbol{M} = \left(\left(\{0\}_{K_6}, \left\{\{K_7\}_{K_1}\right\}_{K_4}\right), \left(\left(K_2, \left\{\left(\{001\}_{K_3}, \{K_6\}_{K_5}\right)\right\}_{K_5}\right), \{K_5\}_{K_2}\right)\right).$$

With the choice $\boldsymbol{C^1} = \{\boldsymbol{K^5}\}_{\boldsymbol{K_2}}$, we obtain

$$\mathcal{D}_1(M)\big((x_1,x_2),((x_3,x_4),x_5)\big) = \begin{cases} \Big((x_1,x_2),\big((x_3,x_4),(x_3,0,\mathcal{B}(\{K_5\}_{K_2})(\mathcal{D}(x_3,x_5)))\big)\Big) & \text{if } (x_3,x_5) \in \mathrm{Dom}_{\mathcal{D}} \\ ((x_1,x_2),((x_3,x_4),(0,0,0)) & \text{otherwise} \end{cases}$$

The target set of $\boldsymbol{D_0(M)}$ is naturally not $\boldsymbol{\tau_0(M)}$, because instead of the copy of $\overline{\boldsymbol{strings}}$ corresponding to $\boldsymbol{C^1}$ we now have a set of the form $\overline{\boldsymbol{strings}} \times \boldsymbol{0} \times \boldsymbol{\tau_0(C^1_{text})}$. We will call this new set $\boldsymbol{\tau_1(M)}$, and so we extend the definition of $\boldsymbol{\tau_0}$ to higher order, up to $\boldsymbol{\tau_{c(M)}(M)}$.First we need the folowing:

Definition 9.45 (Level *i* Subexpression Occurrences). We will say that a subexpression oc-currence $N \sqsubseteq M$ is level *i* with respect to C_M , and denote this relation by $N \sqsubseteq_i M$, if the occurrence *N* is not in the occurrence C^j whenever $i < j$. Let $sub_i(M)$ denote the set of level *i* subexpression occurrences.

Notice, that the level *i* subexpression occurrences are all those which are revealed once C^1, C^2, ... ,C^i are decrypted.

Definition 9.46 (Associated *i*-Tree). Let us inductively define the *i-tree associated* to a pair of expressions $N \sqsubseteq_i M$, and denote it by $T_i(N; M)$:

$$\boldsymbol{\tau_i(K,M)} ::= \overline{\boldsymbol{strings}}$$
$$\boldsymbol{\tau_i(B,M)} ::= \overline{\boldsymbol{strings}}$$
$$\boldsymbol{\tau_i((M_1,M_2),M)} ::= \boldsymbol{\tau_i(M_1,M) \times \tau_i(M_2,M)}$$
$$\mathcal{T}_i(C^j, M) := \begin{cases} \overline{\mathbf{strings}} \times \{0\} \times \mathcal{T}_i(C^j_{\mathrm{text}}, M) & \text{if } j \le i \\ \overline{\mathbf{strings}} & \text{otherwise} \end{cases}$$
$$\boldsymbol{\tau_{i-1}(\{M'\}_K, M)} ::= \overline{\boldsymbol{strings}} \text{ for K} \notin \text{R-Keys(M)}$$
$$\text{Let } \boldsymbol{\tau_i(M)} := \boldsymbol{\tau_i(M,M)}.$$

Note that we only “open” the encryptions performed with the keys in *R-Keys*(*M*) and at each step *i* we only open the C^j such that $j \le i$.

Fact 9.1. For any expressions M and N, we have that $\boldsymbol{\tau_i}(M) \cap \boldsymbol{\tau_i}(N) = \emptyset$ or $\boldsymbol{\tau_i}(M) = \boldsymbol{\tau_i}(N)$.

Similarly, we need to define $G_i(N; M)$ and $D_i(M)$ for $0 < i \leq c(M)$. The first one projects onto the copy of $\overline{\boldsymbol{strings}}$ in $T_i(M)$ that corresponds to N, and the second maps an element in $\boldsymbol{\tau}_{i-1}(M)$ into $\boldsymbol{\tau}_i(M)$ decrypting the string corresponding to C^j with the appropriate key.

Definition 9.48 (*i*-Get Function). For subexpression occurrences $N \sqsubseteq_i M$, $N' \sqsubseteq_i M$ $(0 \leq i \leq c(M))$ such that N occurs in N', let us define the map *i-get-function associated* to the triple (N, N', M), $G_i(N, N', M) : T_i(N', M) \to T_i(N, M)$ inductively as follows:

$$\boldsymbol{G_i(N, N, M) := id_{\tau_i(N,M)}}$$

$$\mathcal{G}_i(N, (M_1, M_2), M) := \begin{cases} \mathcal{G}_i(N, M_1, M) \circ \pi^1_{\mathcal{T}_i(M_1,M)\times\mathcal{T}_i(M_2,M)} & \text{if } N \sqsubseteq M_1 \\ \mathcal{G}_i(N, M_2, M) \circ \pi^2_{\mathcal{T}_i(M_1,M)\times\mathcal{T}_i(M_2,M)} & \text{otherwise} \end{cases}$$

$$\mathcal{G}_i(N, C^j, M) := \mathcal{G}_i(N, C^j_{\text{text}}, M) \circ \pi^3_{\mathcal{T}_i(C^j_{\text{key}},M)\times\{0\}\times\mathcal{T}_i(C^j_{\text{text}},M)}, \text{ for } j \leq i, N \neq C^j$$

define $G_i(N, M) := G_i(N, M, M)$.

Definition 9.49 (Same Position of Subexpression Occurrences). For two expressions M and N, if $\boldsymbol{\tau_i}(M) = \boldsymbol{\tau_i}(N)$, we say that $M' \in sub_i(M)$ and $N' \in sub_i(M)$ *are in the same position at level i*, if $G_i(M', M) = G_i(N', N)$*:*

Let $\Gamma_i(N, M) : sub_i(M) \to sub_i(N)$ denote the unique bijection such that $G_i(M', M) = G_i(\Gamma_i(N, M)\, M', N)$ for all $M' \in sub_i(N)$.

Let

$$\boldsymbol{\tau^{C^i}_{i-1}(M) = \{t \in \tau_{i-1}(M) \mid \left(G_{i-1}\left(C^i_{key}, M\right)t, G_{i-1}\left(C^i, M\right)t\right) \in Dom_D\}}$$

Definition 9.50 (*i*-Decrypting Function). For expressions $N \sqsubseteq_{i-1} M$ and $1 \leq i \leq c(M)$, let us define the map $D_i(N, M) : \boldsymbol{\tau}_{i-1}(M) \to \boldsymbol{\tau}_i(N, M)$ inductively as follows: Let $t \in \boldsymbol{\tau}_{i-1}(M)$

$$D_i(K, M)t := G_{i-1}(K, M)t$$

$$D_i(B, M)t := G_{i-1}(B, M)t \qquad \text{if } K \notin \text{R-Keys}(M)$$

$$D_i(\{M'\}_K, M)t := G_{i-1}(\{M'\}_K, M)t$$

$$D_i((M_1, M_2), M)t := (D_i(M_1, M)t, D_i(M_2, M)t)$$

$$D_i(C^j, M)t :=$$

$$= \begin{cases} \Big(\mathcal{G}_{i-1}(C^j_{\text{key}}, M)t, 0, \mathcal{D}_i(C^j_{\text{text}}, M)\Big) & \text{if } j < i \\ \Big(\mathcal{G}_{i-1}(C^i_{\text{key}}, M)t, 0, \mathcal{B}(C^i_{\text{text}})\big(\mathcal{D}(\mathcal{G}_{i-1}(C^i_{\text{key}}, M)t, \mathcal{G}_{i-1}(C^i, M)t)\big)\Big), t \in \mathcal{T}^{C^i}_{i-1}(M) \\ (0,0,0) & \text{if } t \notin \mathcal{T}^{C^i}_{i-1}(M) \text{ and } j = i \\ \mathcal{G}_{i-1}(C^j, M)t & \text{if } j > i \end{cases}$$

Let

$$\mathcal{D}(M) := \mathcal{D}_{c(M)}(M) \circ \ldots \circ \mathcal{D}_1(M) \circ \mathcal{B}(M)$$

The composition of functions $D_i(M)$ (in order) decrypt all the ciphers that are encrypted with recoverable keys. At the end, D(M) decrypts all ciphers encrypted with recoverable keys upon an input from sampling $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$.

Example 9.20. In our on-going example,

$$\boldsymbol{M} = \left(\left(\{\mathbf{0}\}_{\boldsymbol{K_6}}, \left\{\{\boldsymbol{K_7}\}_{\boldsymbol{K_1}}\right\}_{\boldsymbol{K_4}}\right), \left(\left(\boldsymbol{K_2}, \left\{\left(\{\mathbf{001}\}_{\boldsymbol{K_3}}, \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}}\right)\right\}_{\boldsymbol{K_5}}\right), \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}\right)\right),$$

If y is a sample from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$, then $D(M)y$ has the form

$$\left(\left((\boldsymbol{y_6}, \mathbf{0}, \mathbf{0}), \boldsymbol{y_1}\right), \left(\left(\boldsymbol{y_2}, \left(\boldsymbol{y_5}, \mathbf{0}, \left(\boldsymbol{y_3}, (\boldsymbol{y_5}, \mathbf{0}, \boldsymbol{y_6})\right)\right), (\boldsymbol{y_2}, \mathbf{0}, \boldsymbol{y_5})\right)\right),$$

Where y_2, y_5, y_6 are outcomes of the key-generation algorithms $\boldsymbol{K}_{\boldsymbol{\Phi}(\mathbf{K_2})}, \boldsymbol{K}_{\boldsymbol{\Phi}(\mathbf{K_5})}, \boldsymbol{K}_{\boldsymbol{\Phi}(\mathbf{K_6})}$, respec-tively, y_1 is an undecryptable sample element from $\left[\!\left[\{\{\boldsymbol{K_7}\}_{\boldsymbol{K_1}}\}_{\boldsymbol{K_4}}\right]\!\right]_{\boldsymbol{\Phi}}$, and y_3 is an undecryptable sample from $\left[\!\left[\{\mathbf{001}\}_{\mathbf{K_3}}\right]\!\right]_{\boldsymbol{\Phi}}$. Moreover, ($y_6$, 0, 0) indicates that the key y_6 encrypts the plaintext 0, (y_2, 0, y_5) indicates that the key y_2 encrypts the plaintext y_5 (which is also a key), and so on.

The following lemma essentially claims that if the interpretation is such that conditions (i) and (ii) below hold, then for any two valid expressions M and N, the distribution of $D(M)x$, where x is sampled from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ (let $D(M)([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}})$ denote this distribution), is indistinguishable from the distribution of $D(N)y$, where y is sampled from $[\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$ whenever $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$.

For a function f on $\overline{\boldsymbol{strings}}$, let $f([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}})$ denote the probability distribution of $f(x)$ as x is sampled from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$.

Lemma 9.1. *Let* $\Delta = (Exp_V, \equiv_{\boldsymbol{K}}, \equiv_{\boldsymbol{C}})$ *be a formal logic for symmetric encryption, and let* Φ *be an interpretation of* Exp_V *in* $\Pi = (\{\boldsymbol{K_i}\}_{\boldsymbol{i}\in\boldsymbol{I}}, \boldsymbol{E}, \boldsymbol{D}, \approx)$. *Suppose that this realisation satisfies the following properties for any K, K', K'' ∈ Keys, B ∈ Blocks, M, M', N ∈* Exp_V*:*

(i) *no pair of* $[\![\boldsymbol{K}]\!]_{\boldsymbol{\Phi}}, [\![\boldsymbol{B}]\!]_{\boldsymbol{\Phi}}, [\![(\boldsymbol{M},\boldsymbol{N})]\!]_{\boldsymbol{\Phi}}, [\![\{\boldsymbol{M'}\}_{\boldsymbol{K'}}]\!]_{\boldsymbol{\Phi}}$ *are equivalent with respect to* $\approx$*; that is, keys, blocks, pairs, ciphers are distinguishable.*

(ii) *If* $[\![(\boldsymbol{K},\{\boldsymbol{M}\}_{\boldsymbol{K}}]\!]_{\boldsymbol{\Phi}} \approx [\![(\boldsymbol{K''},\{\boldsymbol{M'}\}_{\boldsymbol{K'}}]\!]_{\boldsymbol{\Phi}}$, *then K' = K''.*

Let M and N be valid formal expressions. Let $\boldsymbol{C_M} = \{\boldsymbol{C_M^1}, \ldots \boldsymbol{C_M^{c(M)}}\}$ *be an enumeration of all ciphers encrypted by recoverable keys in M such that they can be decrypted in this order. Then,* $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$ *implies that c(M) = c(N), and* $\boldsymbol{C_N} = \{\boldsymbol{C_N^1}, \ldots \boldsymbol{C_N^{c(N)}}\}$ *can be enumerated in the order of decryption such that* $\Gamma_{\boldsymbol{c(M)}}(\boldsymbol{N},\boldsymbol{M})\boldsymbol{C_M^i} = \boldsymbol{C_N^i}$. *Moreover, with this enumeration of* C_N , $D_i(M) = D_i(N)$, *and*

$$\boldsymbol{D}(\boldsymbol{M})([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}) \approx \boldsymbol{D}(\boldsymbol{N})([\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}})$$

Proof. Let M and N be expressions such that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$. Since we assumed condition (i) and since the equivalence ≈ is assumed to be invariant under depairing, the pairs that are not encrypted in M and in N must be in the same positions, and so B(M) = B(N) must hold. Since the blow-up function is obtained by repeated application of the inverse of the pairing function, projecting and coupling,

$$\boldsymbol{B}(\boldsymbol{M})([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}) \approx \boldsymbol{B}(\boldsymbol{N})([\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}) \qquad (2.8)$$

As mentioned in Proposition 9.6, if x is sampled from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$, then B(M)x ∈ $\boldsymbol{\tau_0}$(M). Therefore,

$$\boldsymbol{\tau_0}(\mathrm{M}) = \boldsymbol{\tau_0}(\mathrm{N}).$$

Since $\boldsymbol{\tau_0}$(M) = $\boldsymbol{\tau_0}$(N), there is a unique bijection

$$\Gamma_0(\mathrm{N}, \mathrm{M}) : \mathrm{sub}_0(\mathrm{M}) \to \mathrm{sub}_0(\mathrm{N})$$

that satisfies

$$\mathrm{G}_0(\mathrm{M}', \mathrm{M}) = \mathrm{G}_0(\Gamma_0(\mathrm{N}, \mathrm{M})\mathrm{M}', \mathrm{N}):$$

Let $\boldsymbol{C_M^1 = \{C_{M,text}^1\}_{C_{M,key}^1}}$ and $\boldsymbol{L_1 \coloneqq \Gamma_0(N,M)C_{M,key}^1 . L_1}$ must be a key for the following reason:

$$\left(\boldsymbol{G_0\left(C_{M,key}^1, M\right)o\ B(M)}\right)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}}) \approx$$
$$\left(\boldsymbol{G_0\left(C_{M,key}^1, M\right)oB(M)}\right)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}),$$

since we again apply the same function, $\boldsymbol{G_0\left(C_{M,key}^1, M\right)o\ B(M)}$ on $\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}}$ and $\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}$, and this function is made up of depairing, projecting and coupling. But, for the left hand side we clearly have

$$\left(\boldsymbol{G_0\left(C_{M,key}^1, M\right)o\ B(M)}\right)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}}) = \left\llbracket \boldsymbol{C_{M,key}^1} \right\rrbracket_{\boldsymbol{\Phi}},$$

and for the right hand side,

$$\left(\boldsymbol{G_0\left(C_{M,key}^1, M\right)\ o\ B(M)}\right)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}) =$$
$$\left(\boldsymbol{G_0(L_1, N)\ o\ B(N)}\right)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}) = \llbracket \boldsymbol{L_1} \rrbracket_{\boldsymbol{\Phi}}.$$

Therefore, by assumption (i) L_1 must be a key. Similarly,

$$\left(\boldsymbol{G_0\left(C_M^1, M\right)\ o\ B(M)}\right)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}}) \approx \left(\boldsymbol{G_0\left(C_M^1, M\right)\ o\ B(M)}\right)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}})$$

The left-hand side equals $\left\llbracket \boldsymbol{C_M^1} \right\rrbracket_{\boldsymbol{\Phi}}$, hence we need to have an interpretation of a cipher on the right too, implying that for some N' expression and L key,

$$\boldsymbol{\Gamma_0(N,M)C_M^1 = \{N'\}_L}$$

and hence

$$\boldsymbol{G_0\left(C_M^1, M\right) = G_0(\{N'\}_L, N)}. \quad (2.9)$$

Then, according to the foregoing,

$$\left(\left(\boldsymbol{G_0\left(C_{M,key}^1, M\right), G_0\left(C_M^1, M\right)}\right)\boldsymbol{\ o\ B(M)}\right)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})$$
$$\approx \left(\boldsymbol{G_0(L_1, N), G_0(\{N'\}_L, N))\ o\ B(N)}\right)$$

and therefore,

$$\left(\left(\boldsymbol{G_0\left(C_{M,key}^1, M\right), G_0\left(C_M^1, M\right)}\right)\boldsymbol{\ o\ B(M)}\right)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})$$
$$\approx \left(\boldsymbol{(G_0(L_1, N), G_0(\{N'\}_L, N))\ o\ B(N)}\right)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}).$$

But, the left-hand side equals $\left\llbracket \boldsymbol{(C_{M,key}^1, C^1)} \right\rrbracket_{\boldsymbol{\Phi}}$, whereas the right-hand side is $\llbracket \boldsymbol{(L_1, \{N'\}_L)} \rrbracket_{\boldsymbol{\Phi}}$, so we have

$$[\![(\boldsymbol{C^1_{M,key}}, \boldsymbol{C^1_M})]\!]_{\boldsymbol{\Phi}} \approx [\![(\boldsymbol{L_1}, \{\boldsymbol{N'}\}_{\boldsymbol{L}})]\!]_{\boldsymbol{\Phi}}.$$

By assumption (ii) then, L = $\boldsymbol{L_1}$ **follows, because** $\boldsymbol{C^1_M} = \{\boldsymbol{C^1_{M,text}}\}_{\boldsymbol{C^1_{M,key}}}$. But then we can choose the first element of $\boldsymbol{C_N}$ to be the occurrence $\{\boldsymbol{N'}\}_{\boldsymbol{L}}$, and with this choise,

$$\boldsymbol{D_1(M) = D_1(N)}$$

Therefore

$$\boldsymbol{D_1(M)(B(M)([\![M]\!]_{\Phi})) \approx D_1(N)(B(N)([\![N]\!]_{\Phi}))},$$

and therefore,

$$\boldsymbol{\tau}_1(\mathrm{M}) = \boldsymbol{\tau}_1(\mathrm{N});$$

because $\boldsymbol{D_1(M)(B(M)([\![M]\!]_{\Phi}))}$ gives a distribution on $\boldsymbol{\tau}_1$(M), and $\boldsymbol{D_1(N)(B(N)([\![N]\!]_{\Phi}))}$ gives a distribution on $\boldsymbol{\tau}_1$(N).

An argument similar to the one above shows that

$$\mathrm{D}_2(\mathrm{M}) = \mathrm{D}_2(\mathrm{N}):$$

Namely, there is a unique bijection

$$\Gamma_1(\mathrm{N}, \mathrm{M}) : \mathrm{sub}_1(\mathrm{M}) \to \mathrm{sub}_1(\mathrm{N})$$

satisfying

$$\mathrm{G}_1(\mathrm{M}', \mathrm{M}) = \mathrm{G}_1(\Gamma_1(\mathrm{N}, \mathrm{M})\mathrm{M}', \mathrm{N}):$$

Then, just as we proved for L_1, $\mathrm{L}_2 := \Gamma_1(\mathrm{N}, \mathrm{M})\boldsymbol{C^2_{key}}$ must be a key, and

$$\Gamma_1(\mathrm{N}, \mathrm{M})\mathrm{C}^2 = \{\mathrm{N}''\}_{\mathrm{L}2}$$

for some N" expression, implying that

$$\mathrm{D}_2(\mathrm{M}) = \mathrm{D}_2(\mathrm{N}):$$

And so on. So

$$\boldsymbol{D_{c(M)}(M)\ o\ \ldots o\ D_1(M)\big(B(M)([\![M]\!]_{\Phi})\big)}$$
$$\boldsymbol{\approx D_{c(M)}(N) o\ \ldots o\ D_1(N)(B(N)([\![N]\!]_{\Phi}))}$$

since the functions applied on $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ and $[\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$are the same, and they are made up only of depairing, projecting, coupling and decrypting. Then, c(M) ≤ c(N). Reversing the role of M and N in the argument, we get that c(N) ≤ c(M), and so c(M) = c(N). Hence,

$$\mathrm{D(M) = D(N)},$$

and

$$\mathrm{D(M)}([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}) = \mathrm{D(N)}([\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}).$$

Let us illustrate the proof with the following example:

Example 9.21. Suppose again, that

$$\boldsymbol{M} = \left(\left(\{0\}_{K_6}, \left\{\{K_7\}_{K_1}\right\}_{K_4}\right), \left(\left(K_2, \left\{\left(\{001\}_{K_3}, \{K_6\}_{K_5}\right)\right\}_{K_5}\right), \{K_5\}_{K_2}\right)\right),$$

and assume that conditions (i) and (ii) of the lemma are satisfied. Suppose that N is also a valid expression such that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$. Let

$$\boldsymbol{C_M^1} = \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}$$
$$\boldsymbol{C_M^2} = \{(\{\boldsymbol{001}\}_{\boldsymbol{K_3}}, \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}})\}_{\boldsymbol{K_5}}$$
$$\boldsymbol{C_M^3} = \{\boldsymbol{K_6}\}_{\boldsymbol{K_5}}$$
$$\boldsymbol{C_M^4} = \{\boldsymbol{0}\}_{\boldsymbol{K_6}}.$$

M is a pair of two expressions: M = (M_1, M_2). Then, since $[\![(\boldsymbol{M_1}, \boldsymbol{M_2})]\!]_{\boldsymbol{\Phi}} = [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$, condition (i) of the lemma ensures that N must be a pairtoo: N = (N_1; N_2). Then, since

$$[\![\boldsymbol{M_1}]\!]_{\boldsymbol{\Phi}} = \boldsymbol{\pi}^{\boldsymbol{1}}_{\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}} \, \boldsymbol{o} \, [\bullet, \bullet]^{-\boldsymbol{1}}([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}),$$

and

$$[\![\boldsymbol{N_1}]\!]_{\boldsymbol{\Phi}} = \boldsymbol{\pi}^{\boldsymbol{1}}_{\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}} \, \boldsymbol{o} \, [\bullet, \bullet]^{-\boldsymbol{1}}([\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}),$$

(where $\boldsymbol{\pi}^{\boldsymbol{1}}_{\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}}$ denotes projection onto the first component of $\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}$), and since ≈ is assumed to be preserved by depairing and projecting, it follows that

$$[\![\boldsymbol{M_1}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N_1}]\!]_{\boldsymbol{\Phi}}$$

Therefore, since M_1 is a pair, N_1 must be a pair too. Let us recursively apply this argument and this way let us conclude, that the non-encrypted pairs in M are in the same position as the non-encrypted pairs in N, hence

$$B(M) = B(N).$$

It also follows then, that

$$\boldsymbol{\tau_0(M)} = (\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \times ((\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \times \overline{\boldsymbol{strings}}) = \boldsymbol{\tau_0(N)}$$

At this point, we know that N has the form

$$\boldsymbol{N} = ((\boldsymbol{N_3}, \boldsymbol{N_4}), ((\boldsymbol{N_5}, \boldsymbol{N_6}), \boldsymbol{N_7}))$$

Now, we took $\boldsymbol{C_M^1}$ to be $\{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}$, the corresponding string, which is a cipher, is located in the last component of $\boldsymbol{\tau}_0$(M). The key string that decrypts this cipher is located in the third component of $\boldsymbol{\tau}_0$(M). Hence

$$\boldsymbol{G_0}\left(\boldsymbol{C_M^1}, \boldsymbol{M}\right) = \boldsymbol{\pi}^{\boldsymbol{5}}_{\boldsymbol{\tau_0(M)}}$$

and

$$\boldsymbol{G_0}\left(\boldsymbol{C^1_{M,key}}, \boldsymbol{M}\right) = \boldsymbol{\pi}^{\boldsymbol{3}}_{\boldsymbol{\tau_0(M)}}$$

But then, since $\boldsymbol{\pi}^{\boldsymbol{i}}_{\boldsymbol{\tau_0(M)}}$ preserves ≈, it follows that

$$\boldsymbol{\pi}^{\boldsymbol{3}}_{\boldsymbol{\tau_0(M)}}(\boldsymbol{B(M)}(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})) \approx \boldsymbol{\pi}^{\boldsymbol{3}}_{\boldsymbol{\tau_0(M)}}(\boldsymbol{B(N)}(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}))$$

and

$$\boldsymbol{\pi}^{\boldsymbol{5}}_{\boldsymbol{\tau_0(M)}}(\boldsymbol{B(M)}(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})) \approx \boldsymbol{\pi}^{\boldsymbol{5}}_{\boldsymbol{\tau_0(M)}}(\boldsymbol{B(N)}(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}))$$

It is also true that

$$\boldsymbol{\pi}^{\boldsymbol{3}}_{\boldsymbol{\tau_0(N)}} = \boldsymbol{G_0}(\boldsymbol{N_5}, \boldsymbol{N})$$

But

$$\boldsymbol{G_0}\left(\boldsymbol{C^1_{M,key}}, \boldsymbol{M}\right)\left(\boldsymbol{B(M)}(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})\right) = \llbracket \boldsymbol{K_2} \rrbracket_{\boldsymbol{\Phi}},$$

and

$$\boldsymbol{G_0}(\boldsymbol{N_5}, \boldsymbol{N})\left(\boldsymbol{B(N)}(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}})\right) = \llbracket \boldsymbol{K_5} \rrbracket_{\boldsymbol{\Phi}},$$

so

$$\llbracket \boldsymbol{N_5} \rrbracket_{\boldsymbol{\Phi}} \approx \llbracket \boldsymbol{K_2} \rrbracket_{\boldsymbol{\Phi}},$$

and hence, by the assumption (i) of the lemma, it follows that $\boldsymbol{N_5}$ must also be a key, let us denote it with $\boldsymbol{L_1}$. Similary,

$$\boldsymbol{\pi}^{\boldsymbol{5}}_{\boldsymbol{\tau_0(N)}} = \boldsymbol{G_0}(\boldsymbol{N_7}, \boldsymbol{N})$$

But then

$$\llbracket \boldsymbol{N_7} \rrbracket_{\boldsymbol{\Phi}} \approx \left[\!\left[\{\boldsymbol{K_5}\}_{\boldsymbol{K_2}} \right]\!\right]_{\boldsymbol{\Phi}}$$

and therefore N_7 must be a cipher: $N_7 = \{\boldsymbol{N'}\}_{\boldsymbol{L}}$ for some expression N' and key L. To get that $L = L_1$, consider

$$(\boldsymbol{\pi}^{\boldsymbol{3}}_{\boldsymbol{\tau_0(M)}}, \boldsymbol{\pi}^{\boldsymbol{5}}_{\boldsymbol{\tau_0(M)}})\ \boldsymbol{o}\ \boldsymbol{B(M)}(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}}) = \left[\!\left[(\boldsymbol{K_2}, \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}}) \right]\!\right]_{\boldsymbol{\Phi}}$$

and

$$(\boldsymbol{\pi}^{\boldsymbol{3}}_{\boldsymbol{\tau_0(M)}}, \boldsymbol{\pi}^{\boldsymbol{5}}_{\boldsymbol{\tau_0(M)}})\ \boldsymbol{o}\ \boldsymbol{B(N)}(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}}) = \llbracket (\boldsymbol{L_2}, \{\boldsymbol{N'}\}_{\boldsymbol{L}}) \rrbracket_{\boldsymbol{\Phi}}.$$

From this, since the left-hand sides are equivalent, we conclude that $[\![(\boldsymbol{K_2}, \{\boldsymbol{K_5}\}_{\boldsymbol{K_2}})]\!]_{\boldsymbol{\Phi}} \approx [\![(\boldsymbol{L_1}, \{\boldsymbol{N'}\}_{\boldsymbol{L}})]\!]_{\boldsymbol{\Phi}}$, which means by condition (ii) of the lemma that L = L_1:

Therefore, if we define $\boldsymbol{C_N^1}$ as $\{\boldsymbol{N'}\}_{\boldsymbol{L}}$, then these terms and the keys that decrypt them are also in same position, so

$$\boldsymbol{D_1(M) = D_1(N)}$$

Remember from example 9.19, that D_1(M) = D_1(N) does the following:

$$\mathcal{D}_1(M)\big((x_1,x_2),((x_3,x_4),x_5)\big) = \begin{cases} \Big((x_1,x_2),\big((x_3,x_4),(x_3,0,\mathcal{B}(\{K_5\}_{K_2})(\mathcal{D}(x_3,x_5)))\big)\Big) \\ \qquad\qquad \text{if } (x_3,x_5) \in \mathrm{Dom}_{\mathcal{D}} \\ \big((x_1,x_2),((x_3,x_4),(0,0,0))\big) \quad \text{otherwise,} \end{cases}$$

so if x is sampled from $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}$ or $[\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$, then D_1(M)(B(M)x) = D_1(N)(B(N)x) has the form

$$\big((\boldsymbol{x_1}, \boldsymbol{x_2}), \big((\boldsymbol{x_3}, \boldsymbol{x_4}), (\boldsymbol{x_3}, \boldsymbol{0}, \boldsymbol{x_6})\big)\big),$$

and

$$\begin{aligned} &\boldsymbol{\tau_1(M) = \tau_1(N)} \\ &= (\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \times ((\overline{\boldsymbol{strings}} \times \overline{\boldsymbol{strings}}) \\ &\times (\overline{\boldsymbol{strings}} \times \{\boldsymbol{0}\} \times \overline{\boldsymbol{strings}})) \end{aligned}$$

Then, we continue this process until we show that D_4(M) = D_4(N).

Completeness

Let us present the completeness result. Condition (ii) is equivalent to what the authors in [6] call weak key-authenticity. Observe, that the issue of key-cycles never rise throughout the proof.

The proof consists of two separate parts. In the first, it is shown that conditions (i) and (ii) imply that if *M* and *N* are valid expressions and $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$, then there is a key-renaming $\boldsymbol{\sigma}$, such that apart from the boxes, everything else in the patterns of *M* and $N\boldsymbol{\sigma}$ is the same, and the boxes in the two patterns must be in the same positions. Moreover, condition (iii) implies that picking any two boxes of the pattern of $N\boldsymbol{\sigma}$, there is a key-renaming $\boldsymbol{\sigma}_1$ such that applying it to the indexes of these boxes, we obtain the corresponding boxes in the pattern of *M*. Then the

theorem follows, if we prove that using these pairwise equivalences of the boxes, we can construct a $\boldsymbol{\sigma'}$ that leaves the keys of $N\boldsymbol{\sigma}$ outside the boxes untouched, and it maps the indexes of all the boxes of $N\boldsymbol{\sigma}$ into the indexes of the boxes of *M*.

Theorem 9.19. Let $\Delta = (\mathrm{Exp}_V, \equiv_{\boldsymbol{K}}, \equiv_{\boldsymbol{C}})$ be a formal logic for symmetric encryption, such that $\equiv_{\boldsymbol{C}}$ is proper and that $\equiv_{\boldsymbol{K}}$ and $\equiv_{\boldsymbol{C}}$ are independent. Let Φ be an interpretation in $\Pi = (\{\boldsymbol{K_i}\}_{\boldsymbol{i}\in\boldsymbol{I}}, \boldsymbol{E}, \boldsymbol{D}, \approx)$.

Completeness for Φ *holds, if and only if the following conditions are satis-fied: For any K, K', K'* $\in$ *Keys, B* $\in$ *Blocks, M, M', N* $\in$ Exp_V*,*

(i) *no pair of* $[\![\boldsymbol{K}]\!]_{\boldsymbol{\Phi}}$, $[\![\boldsymbol{B}]\!]_{\boldsymbol{\Phi}}$, $[\![(\boldsymbol{M},\boldsymbol{N})]\!]_{\boldsymbol{\Phi}}$, $[\![\{\boldsymbol{M'}\}_{\boldsymbol{K'}}]\!]_{\boldsymbol{\Phi}}$ *is equivalent with respect to* $\approx$*; that is, keys, blocks, pairs, encryption terms are distinguishable,*

(ii) *if* $[\![(\boldsymbol{K},\{\boldsymbol{M}\}_{\boldsymbol{K}})]\!]_{\boldsymbol{\Phi}} \approx [\![(\boldsymbol{K''},\{\boldsymbol{M'}\}_{\boldsymbol{K'}}]\!]_{\boldsymbol{\Phi}}$*, then K' = K'',*

(iii) *for any two pairs of valid encryption terms* $(\{\boldsymbol{M_1}\}_{\boldsymbol{L_1}},\{\boldsymbol{M_2}\}_{\boldsymbol{L_2}})$ *and* $(\{\boldsymbol{N_1}\}_{\boldsymbol{L_1'}},\{\boldsymbol{N_2}\}_{\boldsymbol{L_2'}})$*, we have that*

$$\left[\!\left[(\{\boldsymbol{M_1}\}_{\boldsymbol{L_1}},\{\boldsymbol{M_2}\}_{\boldsymbol{L_2}})\right]\!\right]_{\boldsymbol{\Phi}} \approx \left[\!\left[(\{\boldsymbol{N_1}\}_{\boldsymbol{L_1'}},\{\boldsymbol{N_2}\}_{\boldsymbol{L_2'}})\right]\!\right]_{\boldsymbol{\Phi}}$$

implies

$$\left((\{\boldsymbol{M_1}\}_{\boldsymbol{L_1}},\{\boldsymbol{M_2}\}_{\boldsymbol{L_2}}) \cong (\{\boldsymbol{N_1}\}_{\boldsymbol{L_1'}},\{\boldsymbol{N_2}\}_{\boldsymbol{L_2'}})\right).$$

Proof. The only if part is trivial. In order to prove the if part, consider two expressions *M* and *N* such that $[\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}} \approx [\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}$. By condition (i) and (ii), Lemma 2.18 is applicable, so, $c(M) = c(N)$,

$$\boldsymbol{D}(\boldsymbol{M})([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}}) \approx \boldsymbol{D}(\boldsymbol{N})([\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}}),$$

and

$$\boldsymbol{\tau}_{\boldsymbol{c}(\boldsymbol{M})}(\boldsymbol{M}) = \boldsymbol{\tau}_{\boldsymbol{c}(\boldsymbol{N})}(\boldsymbol{N})$$

In each entry of $\boldsymbol{\tau}_{\boldsymbol{c}(\boldsymbol{M})}(\boldsymbol{M})$ and $\boldsymbol{\tau}_{\boldsymbol{c}(\boldsymbol{N})}(\boldsymbol{N})$, the distribution corresponds either to the interpretation of a key, or of a block, or of an undecryptable cipher (*i.e.* one that corresponds to a box). Naturally, the same blocks must be in the same positions of $\boldsymbol{\tau}_{\boldsymbol{c}(\boldsymbol{M})}(\boldsymbol{M})$ and $\boldsymbol{\tau}_{\boldsymbol{c}(\boldsymbol{N})}(\boldsymbol{N})$, because the dis-tributions of $D(M)([\![\boldsymbol{M}]\!]_{\boldsymbol{\Phi}})$ and $D(N)([\![\boldsymbol{N}]\!]_{\boldsymbol{\Phi}})$ are indistinguishable, and because of condition (i). Hence, the patterns of *M* and *N* contain the same blocks in the same positions. Moreover, since

$D(M)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})$ and $D(N)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}})$ are indistinguishable, the entries in $\boldsymbol{\tau_{c(M)}(M)}$ and in $\boldsymbol{\tau_{c(N)}(N)}$ containing strings sampled from key generation must be in the same places because of (i) again. Furthermore, the indistinguishability of $\boldsymbol{\tau_{c(M)}(M)}$ and $\boldsymbol{\tau_{c(N)}(N)}$also implies that repetitions of a key generation outcome must occur in the same positions of $\boldsymbol{\tau_{c(M)}(M)}$ and $\boldsymbol{\tau_{c(N)}(N)}$ as well. (This is a consequence of the properties of key-generation in definition 2.29.) Therefore the key symbols in the patterns of *M* and *N* change together, so it is possible to rename the recoverable keys of *N* (with a $\equiv_{\boldsymbol{K}}$ preserving function $\boldsymbol{\sigma}$ so that the keys in the pattern of $N\boldsymbol{\sigma}$ are the same as the keys in the pattern of *M*.

Since the distributions of $D(M)(\llbracket \boldsymbol{M} \rrbracket_{\boldsymbol{\Phi}})$ and $D(N)(\llbracket \boldsymbol{N} \rrbracket_{\boldsymbol{\Phi}})$ are indistinguishable, condition (i) implies that the undecryptable ciphers occur in exactly the same entries in $\boldsymbol{\tau_{c(M)}(M)}$ and $\boldsymbol{\tau_{c(N)}(N)}$. This means, that in the pattern of *M* and *N*, the boxes appear in the same position. This together with the conclusions of the previous paragraph means, that apart from the boxes, everything else in the pattern of *M* and of $N\boldsymbol{\sigma}$ must be the same. By replacing *N* with $N\boldsymbol{\sigma}$, we can assume from now on that the recoverable keys of *N* and *M* are identical, and that the pattern of *M* and *N* are the same outside the boxes. Therefore, we only have to show that there is a key renaming $\boldsymbol{\sigma}$' that carries the boxes of *N* into the boxes of *M* without changing the recoverable keys.

Suppose that there are *l* boxes altogether in the pattern of *M* (and hence in the pattern of *N*). Let $\{\boldsymbol{M_1}\}_{\boldsymbol{L_1}}, \{\boldsymbol{M_2}\}_{\boldsymbol{L_2}}, \dots, \{\boldsymbol{M_l}\}_{\boldsymbol{L_l}}$ be the *l* undecryptable terms in *M* that turn into boxes (in *M*) and $\{\boldsymbol{N_1}\}_{\boldsymbol{L_1'}}, \{\boldsymbol{N_2}\}_{\boldsymbol{L_2'}}, \dots, \{\boldsymbol{N_l}\}_{\boldsymbol{L_l'}}$ the corresponding undecryptable terms in *N*. We denote by $\boldsymbol{\mu_i}$ and $\boldsymbol{\nu_i}$ the equivalence classes of $\{\boldsymbol{M_i}\}_{\boldsymbol{L_i}}$ and $\{\boldsymbol{N_i}\}_{\boldsymbol{L_i'}}$, respectively, with respect to $\equiv_{\boldsymbol{C}}$. Then, as we showed above, we have that for $\boldsymbol{i, j \leq l, i \neq j}$,

$$\left[\!\left[\left(\{\boldsymbol{M_i}\}_{\boldsymbol{L_i}}, \{\boldsymbol{M_j}\}_{\boldsymbol{L_j}}\right)\right]\!\right]_{\boldsymbol{\Phi}} \approx \left[\!\left[\left(\{\boldsymbol{N_i}\}_{\boldsymbol{L_i'}}, \{\boldsymbol{N_j}\}_{\boldsymbol{L_j'}}\right)\right]\!\right]_{\boldsymbol{\Phi}}$$

holds since D(M)($[\![M]\!]_{\Phi}$) and D(N)($[\![N]\!]_{\Phi}$)) are indistinguishable, and thus, by condition (iii),

$$\left(\{M_i\}_{L_i}, \{M_j\}_{L_j}\right) \cong \left(\{N_i\}_{L'_i}, \{N_j\}_{L'_j}\right).$$

So, by definition of ≅ , there exists a key-renaming $\sigma_{i,j}$ such that

$$\left(\sqsubset_{\mu_i}, \sqsubset_{\mu_j}\right) = (\sqsubset_{\sigma_{i,j(v_i)}}, \sqsubset_{\sigma_{i,j(v_j)}}),$$

that is, there exists a key-renaming $\sigma_{i,j}$ such that

$$\mu_i = \sigma_{i,j(v_i)} \text{ and } \mu_i = \sigma_{i,j(v_j)} \qquad (9.10)$$

Consider now the class $C = \{\{N_i\}_{L'_i}\}_{i=1}^{l}$. Since we assumed by hypothesis that $\equiv_C$ is proper, by Proposition 2.13 (using *S* = *R-Keys*(*N*) and noticing that $L'_k \notin$ *R-Keys*(*N*)) we have that for each v_k, equivalence class of $\{N_k\}_{L'_k}$, there is a representative Cv_k such that:

(i) *Keys*(Cv_k) ∩ *R-Keys*(*N*) = ∅,

(ii) $L'_m \not\sqsubseteq C_{v_k}$ for all m ∈ $\{1, 2, \ldots, l\}$.

(iii) if $v_{k_1} \neq v_{k_2}$, $|(v_{k_1})_{key}| \neq \infty$ and $|(v_{k_2})_{key}| \neq \infty$, $\boldsymbol{then}\ \boldsymbol{Keys}\left(C_{v_{k_1}}\right) \cap \boldsymbol{Keys}\left(C_{v_{k_2}}\right) \neq \emptyset$ if and only if $(v_{k_1})_{key} = (v_{k_2})_{key} = \{K\}$ for some key K, and in that case

4. Keys($C_{v_{k_1}}$) ∩ Keys($C_{v_{k_2}}$) = $\{K\}$,
5. $C_{v_{k_1}}$ and $C_{v_{k_2}}$ are both of the form $\{\bullet\}_K$, and
6. K $\not\sqsubseteq$ $C_{v_{k_1}}$, K $\not\sqsubseteq$ $C_{v_{k_2}}$.

(iv) if $v_{k_1} \neq v_{k_2}$ and either $|(v_{k_1})_{key}| = \infty$, or $|(v_{k_2})_{key}| = \infty$then $\boldsymbol{Keys}\left(C_{v_{k_1}}\right) \cap \boldsymbol{Keys}\left(C_{v_{k_2}}\right) \neq \emptyset$

Let us define the key-renaming function τ that leaves the recoverable keys of *M* (and *N*) untouched but that maps the boxes in the pattern of *N* to the corresponding boxes in the pattern of *M*. This definition is done by induction.

Induction Basis: Let us start by defining $\boldsymbol{\tau_2^3}$. Since we assumed that $\equiv_{\boldsymbol{C}}$ and $\equiv_{\boldsymbol{K}}$ are inde-pendent, it is possible to modify $\boldsymbol{\sigma_{1,2}}$ such that the resulting renaming function $\boldsymbol{\tau_2}$ that we get leaves

$$\boldsymbol{S_2 = (\bigcup_{i=3}^{l} Keys(C_{v_i}) \cup R - Keys(N)(Keys(C_{v_1}) \cup Keys(C_{v_2}))}$$

untouched and is such that

$$\boldsymbol{\tau_2(v_1) = \sigma_{1,2}(v_1)} \text{ and } \boldsymbol{\tau_2(v_2) = \sigma_{1,2}(v_2)}$$

If we combine the previous equations with (2.10) we have that

$$\boldsymbol{\tau_2(v_1) = \sigma_{1,2}(v_1) = \mu_1}$$

and

$$\boldsymbol{\tau_2(v_2) = \sigma_{1,2}(v_2) = \mu_2.}$$

Induction Hypothesis: Suppose now that we have defined the keys in $\boldsymbol{¿_k}$ in a such a way that $\boldsymbol{¿_k}$ leaves

$$\boldsymbol{S_k = \left(\bigcup_{i=k+1}^{l} Keys(C_{v_i}) \cup R - Keys(N) \right) \setminus (\bigcup_{i=1}^{k} Keys(C_{v_i})),}$$

untouched and is such that

$$\boldsymbol{\tau_k(v_i) = \mu_i} \text{ for all } \boldsymbol{i \leq k}$$

Inductive Step: There are two cases. First suppose that $\boldsymbol{v_{k+1} = v_i}$ for some $\boldsymbol{i \leq k}$. In this case, we define $\boldsymbol{\tau_{k+1} = \tau_k}$. It is obvious that $\boldsymbol{\tau_{k+1}}$ leaves the keys of

$$\boldsymbol{S_{k+1} = \left(\bigcup_{i=k+2}^{l} Keys(C_{v_i}) \cup R - Keys(N) \right) \setminus (\bigcup_{i=1}^{k+1} Keys(C_{v_i})),}$$

untouched and is such that

$$\boldsymbol{\tau_{k+1}(v_i) = \mu_i} \text{ for all } \boldsymbol{i \leq k + 1}$$

since $\boldsymbol{C_{v_{k+1}} = C_{v_i}}$ and $\boldsymbol{v_{k+1} = v_i}$.

In the other case, suppose that $\boldsymbol{v_{k+1} \neq v_i}$ for all $\boldsymbol{i \leq k}$. Consider now the substitution $\boldsymbol{\sigma_{j,(k+1)}}$ with $\boldsymbol{j \leq k}$. By (9.10) we have that

$$\boldsymbol{\mu_j = \sigma_{j,(k+1)}(v_j)}$$

and

$$\boldsymbol{\mu_{k+1} = \sigma_{j,(k+1)}(v_{k+1})}.$$

Since $\equiv_{\boldsymbol{C}}$ and $\equiv_{\boldsymbol{K}}$ are independent, considering

$$\boldsymbol{S} = \left(\bigcup_{\boldsymbol{i=1}}^{\boldsymbol{l}} \boldsymbol{Keys}\left(\boldsymbol{C_{v_i}}\right) \cup \boldsymbol{\tau_k}\left(\bigcup_{\boldsymbol{i=1}}^{\boldsymbol{l}} \boldsymbol{Keys}\left(\boldsymbol{C_{v_i}}\right) \right) \cup \mathbf{R} - \mathbf{Keys}(\mathbf{N}) \right) \setminus \boldsymbol{Keys}\left(\boldsymbol{C_{v_{k+1}}}\right)$$

And $\boldsymbol{C} = \{\boldsymbol{C_{v_{k+1}}}\}$, we have that it is possible to modify $\boldsymbol{\sigma_{j,(k+1)}}$ to $\boldsymbol{\sigma}^*$ such that

$$\boldsymbol{\sigma}^*(\mathrm{K}) = \mathrm{K} \text{ for all } K \in S$$

and

$$\boldsymbol{\sigma^*(v_{k+1}) = \sigma_{j,(k+1)}(v_{k+1})}$$

Using (9.12), we can rewrite the previous equation as

$$\boldsymbol{\sigma^*(v_{k+1}) = \sigma_{j,(k+1)}(v_{k+1}) = \mu_{k+1}}$$

Thus, we have two substitutions, $\boldsymbol{\tau_k}$ and $\boldsymbol{\sigma}^*$ such that

$$\boldsymbol{\tau_k(v_i) = \mu_i} \text{ for all } \boldsymbol{i \leq k} \qquad (9.13)$$

and

$$\boldsymbol{\sigma^*(v_{k+1}) = \mu_{k+1}} \qquad (9.14)$$

Our goal now is to combine these two substitutions into one substitution $\boldsymbol{\tau_{k+1}}$ such that

$$\boldsymbol{\tau_{k+1}(v_i) = \mu_i} \text{ for all } \boldsymbol{i \leq k+1} \quad (9.15)$$

and that leaves untouched the keys in

$$\boldsymbol{S_{k+1}} = \left(\bigcup_{\boldsymbol{i=k+2}}^{\boldsymbol{l}} \boldsymbol{Keys}\left(\boldsymbol{C_{v_i}}\right) \cup \boldsymbol{R - Keys(N)} \right) \setminus \left(\bigcup_{\boldsymbol{i=1}}^{\boldsymbol{k+1}} \boldsymbol{Keys}\left(\boldsymbol{C_{v_i}}\right) \right),$$

We can immediately notice that by definition, $\boldsymbol{\tau_k}$ only changes the keys in $(\bigcup_{\boldsymbol{i=1}}^{\boldsymbol{k}} \boldsymbol{Keys}(\boldsymbol{C_{v_i}}))$ (recall (9.11)) and that $\boldsymbol{\sigma}^*$ only alters the keys in $\mathrm{Keys}(\boldsymbol{C_{v_{k+1}}})$, thus ensuring (9.16). We also notice that from (9.13) and (9.14), (9.15) follows. So, if it is possible to "merge" the two substi-tutions, the result follows. We do this by showing that the two substitutions are compatible. We show that if both substitutions change the value of one key *K*, then they change it to the same value, that is,

we show that if for a key K, $\boldsymbol{\tau_k(K) \neq K}$ and $\boldsymbol{\sigma^*(K) \neq K}$ then $\boldsymbol{\tau_k(K) = \sigma^*(K)}$.

Suppose that both $\boldsymbol{\tau_k}$ and $\boldsymbol{\sigma^*}$ change the value of a key *K'*. Then, by the definition of the two substitutions,

$$\boldsymbol{K'\left(\bigcup_{i=1}^{k} Keys(C_{v_i})\right) \cap Keys(C_{v_{k+1}})}.$$

that is

$$\boldsymbol{K' \in Keys(C_{v_i}) \bigcap \mathbf{Keys}(C_{v_i})}$$

for some i $\in \{\boldsymbol{1, \dots, k}\}$. By the way we constructed the representatives $\boldsymbol{C_{v_k}}$ we have that foe any two different equivalence classes $\boldsymbol{v_{k_1}}$ and $\boldsymbol{v_{k_2}}$, $\boldsymbol{Keys(C_{v_i}) \cap Keys(C_{v_i}) = \emptyset}$ (whenever $\boldsymbol{|(v_i)key| = \infty}$ or $\boldsymbol{|(v_{k+1})key| = \infty}$) or $\boldsymbol{Keys\left(C_{v_{k_1}}\right) \cap Keys\left(C_{v_{k_2}}\right) \neq \emptyset}$ if and only if $\boldsymbol{(v_{k_1})_{key} = (v_{k_2})_{key} = \{K\}}$, and in that case

$$\boldsymbol{Keys\left(C_{v_{k_1}}\right) \cap Keys\left(C_{v_{k_2}}\right) = \{K\}}.$$

Since $\boldsymbol{\{N_i\}_{L_i'} \in v_i}$ and $\boldsymbol{\{N_{k+1}\}_{L_{k+1}'} \in v_{k+1}}$, we have that $\boldsymbol{L_i' \in (v_i)_{key}}$ and $\boldsymbol{L_{k+1}' \in (v_{k+1})_{key}}$, and using (2.18) it follows that

$$\boldsymbol{K' = L_i' = L_{k+1}'}.$$

We just proved that the only key that both $\boldsymbol{\tau_k}$ and $\boldsymbol{\sigma^*}$ change at the same time is *K'* so we just need to prove that they change it to the same value (in order to be compatible), that is,

$$\boldsymbol{\tau_k(K') = \sigma^*(K')}.$$

By (9.18) we have that $\boldsymbol{\left|(v_{k+1})_{key}\right| = 1}$ and so, using Proposition 9.12 and (9.14) it follows that

$$\boldsymbol{\left|(\mu_{k+1})_{key}\right| = \left|\sigma^*((v_{k+1})_{key})\right| = 1}\ .$$

Since $\boldsymbol{\{M_{k+1}\}_{L_{k+1}} \in \mu_{k+1}}$, we have that $\boldsymbol{L_{k+1} \in (\mu_{k+1})_{key}}$. So, from the previous equation and (9.18) it follows that

$$\boldsymbol{\sigma^*(K') = \sigma^*(L_{k+1}') = L_{k+1}} \qquad (9.20)$$

If we apply the same reasoning to $\boldsymbol{v_i}$ and $\boldsymbol{\tau_k}$, again by (9.18) we have that $\left|(\boldsymbol{v_i})_{\boldsymbol{key}}\right| = \boldsymbol{1}$ and so, using Proposition 9.12 and (9.13) it follows that

$$\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \left|\boldsymbol{\tau_k}((\boldsymbol{v_i})_{\boldsymbol{key}})\right| = \boldsymbol{1}.$$

Since $\{\boldsymbol{M_i}\}_{\boldsymbol{L_i'}} \in \boldsymbol{\mu_i}$, we have that $\boldsymbol{L_i} \in (\boldsymbol{\mu_i})_{\boldsymbol{key}}$. So, from the previous equation and (2.18) it follows that

$$\boldsymbol{\tau_k}(\boldsymbol{K'}) = \boldsymbol{\tau_k}(\boldsymbol{L_i'}) = \boldsymbol{L_i} \qquad (9.21)$$

Now consider the substitution $\boldsymbol{\sigma_{i,(k+1)}}$. By (9.10) we have that

$$\boldsymbol{\mu_i} = \boldsymbol{\sigma_{i,(k+1)}}(\boldsymbol{v_i}) \text{ and } \boldsymbol{\mu_{k+1}} = \boldsymbol{\sigma_{i,(k+1)}}(\boldsymbol{v_{k+1}}).$$

Using (9.18) and Proposition 9.2 it follows that

$$\left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \left|\boldsymbol{\sigma_{i,(k+1)}}((\boldsymbol{v_i})_{\boldsymbol{key}})\right| = 1 \text{ and } \left|(\boldsymbol{\mu_i})_{\boldsymbol{key}}\right| = \left|\boldsymbol{\sigma_{i,(k+1)}}((\boldsymbol{v_{k+1}})_{\boldsymbol{key}})\right| = 1$$

As said before, $\boldsymbol{L_i} \in (\boldsymbol{\mu_i})_{\boldsymbol{key}}, \boldsymbol{L_i'} \in (\boldsymbol{v_i})_{\boldsymbol{key}}$, $\boldsymbol{L_{k+1}} \in (\boldsymbol{\mu_{k+1}})_{\boldsymbol{key}}$, and $\boldsymbol{L_{k+1}'} \in (\boldsymbol{v_{k+1}})_{\boldsymbol{key}}$ and so

$$\boldsymbol{\sigma_{i,(k+1)}}(\boldsymbol{L_i'}) = \boldsymbol{L_i} \boldsymbol{\ and\ } \boldsymbol{\sigma_{i,(k+1)}}(\boldsymbol{L_{k+1}'}) = \boldsymbol{L_{k+1}}$$

Combining this with (9.19), since $\boldsymbol{L_i'} = \boldsymbol{L_{k+1}'}$, we have that

$$\boldsymbol{L_i} = \boldsymbol{L_{k+1}}$$

and so by (9.21), (9.22), and (9.20)

$$\boldsymbol{\tau_k}(\boldsymbol{K'}) = \boldsymbol{L_i} = \boldsymbol{L_{k+1}} = \boldsymbol{\sigma^*}(\boldsymbol{K'}).$$

Thus for any key *K'* such that both $\boldsymbol{\tau_k}$ and $\boldsymbol{\sigma^*}$ change the value, they are compatible. We then define $\boldsymbol{\tau_{k+1}}$ as *i\;* S

$$\boldsymbol{\tau_{k+1}}(\boldsymbol{K}) = \begin{cases} \boldsymbol{\sigma^*}(\boldsymbol{K}) \ \boldsymbol{if\ K} \in \boldsymbol{Keys}\left(\boldsymbol{C_{v_{k+1}}}\right) \\ \boldsymbol{\tau_k}(\boldsymbol{K}) \ \boldsymbol{otherwise} \end{cases},$$

Note that by definition og $\boldsymbol{\tau_l}$, it does not change the keys in $\boldsymbol{S_l} = \boldsymbol{R} - \boldsymbol{Keys}(\boldsymbol{N}) \backslash (\cup_{\boldsymbol{i=1}}^{\boldsymbol{l}} \boldsymbol{Keys}\,(\boldsymbol{C_{v_i}}))$ but, by properness, we have that $\boldsymbol{Keys}\left(\boldsymbol{C_{v_i}}\right)\boldsymbol{R} - \boldsymbol{Keys}(\boldsymbol{N}) = \emptyset$ for all $\boldsymbol{1} \leq \boldsymbol{i} \leq \boldsymbol{l}$ which implies that $\boldsymbol{\tau_l}$ does not the keys in R-Keys(N).

The substitution $\boldsymbol{\tau}$ that satisfies the required properties, i.e., that leaves the recoverable keys of *M* and *N* untouched, but maps the boxes of the pattern of *N* into the corresponding boxes in the pattern of *M*, is

defined as $\boldsymbol{\tau_l}$ (l is the number of boxes in the pattern of M) and that is what we needed to complete the proof.

Remark 9.4. Observe, that condition (iii) of the theorem is trivially satisfied when there is only one box, that is, when all encryption terms are equivalent under $\equiv_C$. Also, if completeness holds for a certain choice of $\equiv_C$, then, if $\equiv'_C$ is such that $M \equiv_C N$ implies $M \equiv'_C N$—i.e. when $\equiv'_C$ results fewer boxes—then completeness holds for $\equiv'_C$ as well. Therefore, we can say, that the key to completeness is not to have too many boxes.

Example 9.22 (Completeness for Type-1 and Type-2 Encryption Schemes). The complete-ness part of our earlier theorems for type-1 and type-2 encryption schemes are clearly special cases of this theorem, because the formal language we introduced for these schemes were such that $\equiv_C$ is proper and $\equiv_K$ and $\equiv_C$ are independent, and the conditions of the theorems are anal-ogous.

Example 9.23 (Completeness for One-Time Pad). The formal logic for OTP is such that $\equiv_C$ is proper and $\equiv_K$ and $\equiv_C$ are independent. Furthermore, con-dition (i) of. Condition (ii) is also satisfied because of the tagging: the reason ultimately is that decrypting with the wrong key will sometimes result invalid endings. Condition (iii) is also satisfied, since the pairs of encryption terms must be encrypted with different keys (in OTP, we cannot use the keys twice), and the equivalence $[\![(\{M_1\}_{L_1}, \{M_2\}_{L_2})]\!]_\Phi \approx [\![(\{N_1\}_{L'_1}, \{N_2\}_{L'_2})]\!]_\Phi$ implies that the corresponding lengths in the two encryption terms wust be the same: $l(\{M_1\}_{L_1}) = l(\{N_1\}_{L'_1})$ and $l(\{M_2\}_{L_2}) = l(\{N_2\}_{L'_2})$ which implies $(\sqsubset_{l(\{M_1\}_{L_1})}, \sqsubset_{(\{M_2\}_{L_2})}) = (\sqsubset_{l(\{N_1\}_{L'_1})}, \sqsubset_{l(\{N_2\}_{L'_2})})$. Therefore, $(\{M_1\}_{L_1}, \{M_2\}_{L_2}) \cong (\{N_1\}_{L'_1}, \{N_2\}_{L'_2})$. In conclusion, the formal logic is complete.

Advancement questions

1. What are the main conditions of the Abadi-Rogaway Soundness Theorem?
2. What are the main features of the AR Equivalence of Formal Expressions?
3. Through what the equivalence of expressions is able to be obtained?
4. What does KDM security mean?
5. Why does the soundness in the presence of key-cycles is not possible to prove with the security notion adopted by Abadi and Rogaway?
6. Why does type-0 security is not strong enough to ensure soundness in the case of key-cycles?
7. What are the features of the 'type-1' encryption schemes?
8. What does type-1 encryption in the terminology of Abadi and Rogaway mean?
9. What is the difference between the type-1 and type-2 encryptions?
10. Why does it possible to construct encryption schemes that are type-0, but fail to provide soundness in the presence of key-cycles?

REFERENCES

[1] Pedro Miguel dos Santos Alves Madeira Adao Formal methods for the analysis of security protocols / Pedro Miguel dos Santos Alves Madeira Adao // PhD thesis . – 2006. – Universidad et Tecnica de Lisboa Instituto Superior Tecnico.

[2] M. Abadi Reconciling two views of cryptography (the computational soundness of formal encryption) / M. Abadi , P. Rogaway // Journal of Cryptology. – 2002. - 15(2). Pp.103–127.

[3] J. Black Encryption-scheme security in the presence of key-dependent messages / J. Black, P. Rogaway, T. Shrimpton // Lecture Notes in Computer Science. – 2002 - Volume 259 – pp.62–75.

[4] M. Abadi Reconciling two views of cryptography / M. Abadi, P. Rogaway // Lecture Notes in Computer Scien. – 2000. – vol. 1872. – pp. 3–22.

[5] J. Camenisch An efficient system for non-transferable anonymous credentials with optional anonymity revocation / J. Camenisch and A. Lysyanskaya // Lecture Notes in Computer Science. – 2001. - volume 2045. – pp. 98–118.

[6] O. Horvitz Weak key authenticity and the computational completeness of formal encryption / O. Horvitz and V. Gligor // Boneh. - pages 530–547.

[7] A. C. Yao. Theory and applications of trapdoor functions / A. C. Yao // In 23rd IEEE Symposium on Foundations of Computer Science (FOCS). – 1982 . – pp. 80–91.

PART 3. FORMAL AND INTELLECTUAL METHODS FOR SYSTEM SECURITY AND RESILIENCE

Content of the PART3

GLOSSARY

PPT - probabilistic polynomial-time
QRAM - a quantum random access machine
QPM - a quanlum polynomil machine
RAM - random access machine
RSA algorithm – Rivest's, Shamir's and Adleman's algorithm
IDS - Intrusion Detection System
AI - Artificial Intelligence
ANN - Artificial Neural Networks
SVM - Support Vector Machines
GA - Genetic Algorithms
FNN - Fuzzy Neural Networks
CI - Computational Intelligence
DM - Data Mining
FS - Fuzzy System
MF - Membership Function
DOS - Denial of Service
R2L - Remote to User
U2R - User to Root
SVMs - Support Vector Machines
SVR - Support Vector Regression
SVs - Support Vectors
GA - Genetic Algorithm
FN - Functional Networks
FNN - a Fuzzy Neural Network
SVM - Support Vector Machines
T2FL - Type-2 Fuzzy Logic
FFS - FN-Fuzzy Logic-SVM
FSF - FN-SVM-Fuzzy Logic
LP - Linear Programming
CS - computer system

CFG - control flow graph
CS - computer system
AMAS - antiviral multi-agent system
DTMC - discrete time Markov chains
CTMC - continuous-time Markov chains
MDP - Markov decision processes
PCTL - Probabilistic Computation Tree Logic
CSL - Continuous Stochastic Logic
WAL - Write-ahead logging

Introduction

Process Algebras have been widely used in the study of security of concurrent systems [1-9]. In spite of their success in proving security of cryptographic protocols, mainly secrecy and authenticity properties, all these are stated in the so called Dolev-Yao Model, hence no real cryptographic guarantees are achieved.

Another approach is to supplement process calculi with concrete probabilistic or polynomi-al-time semantics [18]. Unavoidably, reasoning on processes becomes more difficult.

In this Chapter, we present a process calculus that enjoys both the simplicity of an abstract symbolic model and a concrete (sound and complete) implementation that achieves strong cryp-tographic guarantees. Our calculus is a variant of the pi calculus with high level security prim-itives; it provides name mobility, reliable messaging and authentication primitives, but neither explicit cryptography nor probabilistic behaviours.

Taking advantage of concurrency theory, it supports simple reasoning, based on labelled transitions and observational equivalence. This chapter presents its concrete implementation in a computational setting. The implementation relies on standard cryptographic primitives, compu-tational security definitions (CCA2 for encryption [10], CMA for signing [11], recalled in Appendix A), and networking assumptions. It also combines typical distributed implemen-tation mechanisms such as abstract machines, marshaling and unmarshaling, multiplexing, and basic communications protocols.

We establish general completeness results in the presence of active probabilistic polynomial-time adversaries, for both trace properties and observational equivalences, essentially showing that high level reasoning accounts for all low-level adversaries.

This chapter illustrates the approach by coding security protocols and establishing their computational correctness by simple formal reasoning.

This Chapter is organised as follows: it starts by describing the low-level target model as the constraints imposed by this will drive the design of the high-level language, then high-level language and semantics is presented, and notion of high-level equivalence are defined

and illustrated. Chapter is also devoted to applications. Anonymous forwarders in the language and exhibit an example of an electronic payment protocol are encoded. As an example the encoding of an initialisation protocol is given, that is, given any system *S* that possibly shares names and certificates among principals, we can always find an initial system S^o where principals share no information, such that there is a transition from S^o to *S*. Chapter describes concrete implementation, and results.

10.1 Low-Level Target Model

Before presenting the language design and implementation, let us specify the target systems. Let us do this, as the design of our language is, in part, driven by the target model. We have to be as abstract as possible, but at the same time we need to faithfully abstract the properties of the computational implementation.

As an example, we want our high-level environments to have the same capabilities as the low-level adversaries, that are probabilistic polynomial-time (PPT) cryptographic algorithms. We follow the conservative assumption that an adversary controls all network traffic: it can intercept, delay, or even block permanently any communication between principals. For that, we cannot guarantee message delivery, nor implement private channels that prevent traffic analysis. Reflect-ing this in the high-level semantics implies that the simple pi-calculus rule $\bar{c}\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q\{M/x\}$, which models silent communication is too abstract for our purposes. (Consider *P* and *Q* two processes that are implemented in two separate machines connected by a public net-work, and even if *c* is a restricted channel, the adversary can simply block all communications.)

We consider systems that consist of a finite number of principals $a, b, c, e, u, v, \ldots \in$ Prin. Each principal *a* runs its own program, written in our high-level language and executed by the PPT machine M_a defined in Section 3.5. Each machine M_a has two wires, in_a and out_a, rep-resenting a basic network interface. When activated, the machine reads a bitstring from in_a, performs some local computation, then writes a bitstring on out_a and yields. The machine em-beds probabilistic algorithms for encryption, signing, and random-number generation—thus the machine outputs are random variables. The machine is also

parameterised by a security parameter $\eta \in \mathbb{N}$ —intuitively, the length for all keys—thus these outputs are ensembles of probabilities.

Some of these machines may be corrupted, under the control of the attacker; their implemen-tation is then unspecified and treated as part of the attacker. We let $a, b, c \in H$ with $H \subset$ Prin range over principals that comply with our implementation, and let $M = (M_a)_{a \in H}$ describe the whole system. We denote by e a principal controlled by the adversary ($e \in Prin \backslash H$) and by u, v an arbitrary principal in Prin. Of course, when a interacts with $u \in Prin$, its implementation M_a does not know whether $u \in H$ or not.

The adversary, A, is a PPT algorithm that controls the network, the global scheduler, and some compromised principals. At each moment, only one machine is active: whenever an ad-versary delivers a message to a principal, the machine for this principal is activated, runs until completion, and yields an output to the adversary. We have then the following definition:

Definition 10.1 (Run). We define a *run* of A and M **with** security parameter $\eta \in \mathbb{N}$ as follows:

- key materials, with security parameter η , are generated for every principal $a \in Prin$;
- every M_a is activated with 1^η, the keys for a, and the public keys for all $u \in Prin$;
- A is activated with 1^η, the keys for $e \in Prin \backslash H$, and the public keys for $a \in H$;
- A performs a series of low-level exchanges of the form: f writes a bitstring on wire in_a and activates M_a for some $a \in H$; upon completion of M_a, A reads a bitstring on out_a;
- A returns a bitstring s, written $s \leftarrow A[M]$.

We keep η implicit whenever possible.

At each Step 4, the adversary A can choose a and compute the bitstring written on in_a from any previously-received materials, including principal keys and bitstrings collected from previ-ous exchanges.

By design, our low-level runs do not render attacks based on timed properties, such as for instance any observation of the time it takes for each machine to reply. Although the risk of quantitative traffic analysis may be significant, it can be mitigated independently, for instance by

sending messages according to a fixed schedule. We leave this discussion outside the scope of this dissertation.

To study the security properties of these runs, we compare systems that consist of machines running on behalf of the same principals $H \subseteq$ Prin, but with different internal programs and states. Intuitively, two systems are equivalent when no PPT adversary, starting with the informa-tion normally given to the principals $e \in Prin\backslash H$, can distinguish between their two behaviours, except with negligible probability, Definition A.1. This notion is called *computational indis-tinguishability* and was introduced by Goldwasser and Micali [12]. We state it here in a different but equivalent way.

Definition 10.2 (Low-Level Equivalence). Two systems M^0 and M^1 are indistinguishable, written $M^0 \approx M^1$, if for all PPT adversaries A:

$$|\Pr[1 \leftarrow A[M^0]] - \Pr[1 \leftarrow A[M^1]]| \leq neg(\eta).$$

Our goal is to develop a simpler, higher-level semantics that entails indistinguishability.

10.2 A Distributed Calculus with Principals and Authentica-tion

We now present our high-level language. We successively define terms, patterns, processes, configurations, and systems. We then give their operational semantics. Although some aspects of the design are unusual, the resulting calculus is still reasonably abstract and convenient for distributed programming.

Syntax and Informal Semantics

Definition 10.3 (Names, Terms, Patterns). Let Prin be a finite set of *principal identities*. Let Name be a countable set of *names* disjoint from Prin. Let f range over a finite number of function symbols, each with a fixed arity $k \geq 0$. We define *terms* and *patterns* by the following grammar:

V, W ::=	Terms
x, y	variable
m, n ∈ Name	name
a, b, e, u, v ∈ Prin	principal identity

$f(V_1, \ldots, V_k)$	constructed term (when f has arity k)
$T, U ::=$	Patterns
$?x$	variable (binds x)
T *as* $?x$	alias (binds x to the term that matches T)
V	constant pattern
$f(T_1, \ldots, T_k)$	constructed pattern (when f has arity k)

As usual in process calculi, names and principal identities are atoms, which may be compared with one another but otherwise do not have any structure. Constructed terms represent structured data, much like algebraic data types in ML or discriminated unions in C. They can represent constants and tags (when $k = 0$), tuples, and formatted messages. As usual, we write tag and (V_1, V_2) instead of tag() and $\text{pair}(V_1, V_2)$.

Patterns are used for analysing terms and binding selected subterms to variables. For instance, the pattern (tag, $?x$) matches any pair whose first component is tag and binds x to its second component. We write for a variable pattern that binds a fresh variable.

Definition 10.4 (Local Processes). *Local processes* represent the active state of principals, and are defined by the following grammar:

$P, Q, R ::=$	Local processes
V	asynchronous output
$(T).Q$	input (binds $bv(T)$ in Q)
$*(T).Q$	replicated input (binds $bv(T)$ in Q)
match V with T in Q else Q'	matching (binds $bv(T)$ in Q)
$vn.P$	name restriction ("new", binds n in P)
$P \mid P'$	parallel composition
0	inert process

The asynchronous output V is just a pending message; its data structure is explained below. The input $(T).Q$ waits for an output that matches the pattern T then runs process Q with the bound variables of T substituted by the matching subterms of the output message. The replicated input $*(T).Q$ behaves similarly but it can consume any number of outputs that match T and fork a copy of Q for each of them. The match process runs Q if V matches T, and runs Q' otherwise. The name restriction creates a fresh name n then runs P. Parallel composition represents processes that run in parallel, with the inert process 0 as unit.

Free and bound names and variables for terms, patterns, and processes are defined as usual: x is bound in T if $?x$ occurs in T; n is bound in $vn.P$; x is free in T if it occurs in T and is not bound in T . An expression is closed when it has no free variables; it may have free names.

Definition 10.5 (Local Contexts). A *local context* is a process with a hole instead of a subprocess. We say that a context is an *evaluation context* when the hole replaces a subprocess P or P' in the grammar of Definition 3.4. If it replaces a subprocess Q or Q' we call it a *guarded context*.

Our language features two forms of authentication, represented as two constructors auth and cert of arity 3 plus well-formed conditions on their usage in processes.

Definition 10.6 (Authenticated Messages, Certificates). *Authenticated messages* between principals are represented as terms of the form $\text{auth}(V_1,V_2,V_3)$, written $V_1{:}V_2\langle V_3\rangle$ where V_1 is the sender, V_2 the receiver, and V_3 the content. We let M and N range over messages. The message M is *from* a (respēctively *to* a) if a is the sender (respectively the receiver) of M.

Certificates issued by principals are represented as terms of the form $\text{cert}(V_1,V_2,V_3)$, written $V_1\{V_2\}_{V_3}$, where V_1 is the issuer, V_2 the content, and V_3 the label.

Labels in certificates reflect cryptographic signature values in their implementation. They are often unimportant (and omitted), since our processes use a constant label 0 in their certificates and ignore labels (using) in their certificate patterns. Nonetheless, they are necessary because the standard definition of security for signatures (CMA-security, Definition A.6) does not exclude the possibility that the attacker produce different signature values for certificates with identical issuer and content. If we do not include labels in our definition of high-level certificates, we could be excluding attacks.

Example 10.1. Consider a protocol where adversarial principal e receives a certificate $cert_1$ from a, forges a second certificate $cert_2$ using some malleability property of the signing scheme, and then forwards $cert_1$ to b and $cert_2$ to c.

If later e receives $cert_i$ from d, he may discover part of the topology of the network, as $i = 1$ if d is connected to b and $i = 2$ if d is connected

to *c*. If the attack to the protocol depends upon the knowledge of the network, we have an attack.

If we do not account for this possibility in our high-level semantics, that is, use different labels for different certificates, we could never capture this attack as the received certificate by *e* would be equivalent regardless of $i = 0$ or $i = 1$.

Although both authenticated messages and certificates provide some form of authentication, authenticated messages are delivered at most once, to their designated receiver, whereas cer-tificates can be freely copied and forwarded within messages. Hence, certificates conveniently represent transferable credentials and capabilities. They may be used, for instance, to code decentralised access-control mechanisms.

Example 10.2. As an example, $a{:}b\langle \text{Hello} \rangle$ is an authentic message from *a* to *b* with content Hello, a constructor with arity 0, for which *b* (and only *b*) can verify that it is coming from $aa\{b, \text{Hello}\}$ is a certificate signed by *a* with the same subterms that can be sent, received, and verified by any principal.

We let $\emptyset(V)$ be the set of certificates included in *V* and let $\emptyset(V)_X \subseteq \emptyset(V)$ be those certifi-cates issued by $u \in X$. For instance, we have

$$\emptyset(a\{0, b\{1\}, c\{2\}\})_{\{a,b\}} = \{b\{1\}, a\{0, b\{1\}, c\{2\}\}\}$$

Definition 10.7 (Well Formed Process). Let *P* be a local process. We say that *P* is *well-formed for a 2* Prin when:

- any certificate in *P* that includes a variable or a bound name is of the form $a\{V_2\}_0$;
- no pattern in *P* binds any certificate label; and
- no pattern used for input in *P* matches any authenticated message from *a*.

Condition 1 states that the process may produce new certificates only with issuer *a*; in addi-tion, the process may contain previously-received certificates issued by other principals. (We do not restrict certificate patterns—a pattern that tests a certificate not available to *a* will never be matched.) Condition 2 restricts access to labels, so that labels only affect comparisons between certificates. Condition 3 prevents that authenticated messages sent by *P* be read back by some local input.

Finally, we are now able to define configurations and systems. A *configuration* is an assembly of running principals, each with its own local state, plus an abstract record of the messages intercepted by the environment and not forwarded yet to their intended recipients. A *system* is a top-level configuration plus an abstract record of the environment's knowledge, as a set of certificates previously issued and sent to the environment by the principals in *C*.

Definition 10.8 (Configurations, Systems). *Configurations* and *systems* are defined by the fol-lowing grammar:

C ::=	configurations
$a[P_a]$	principal a with local state P_a
M / i	intercepted message M with index i
$C \setminus C'$	distributed parallel composition
$vn.C$	name restriction ("new", binds n in C)
S ::=	systems
$\Phi \vdash C$	configuration C exporting certificates Φ

and satisfy the following well-formed conditions:

- In configurations, intercepted messages have distinct indices i and closed content M; principals have distinct identities a and well-formed local processes P_a.
- In systems, let H be the set of identities for all defined principals, called *compliant prin-cipals*; intercepted messages are from a to b for some $a, b \in H$ with $a \neq b$; Φ is a set of closed certificates with label 0 such that $\emptyset(\Phi)_H = \Phi$.

Operational Semantics

We define our high-level semantics in two stages: local reductions between processes, then global labelled transitions between systems and their (adverse) environment. Processes, configurations, and systems are considered up to renaming of bound names and variables.

Local Reductions

We start by defining *structural equivalence*. It represents structural rearrangements for local processes. Intuitively, these rearrangements are not observable (although this is quite hard to implement).

Definition 10.9 (Structural Equivalence for Processes). *Structural equivalence*, written $P \equiv P'$, is defined as the smallest congruence such that:

$$P \equiv P|0$$
$$P|Q \equiv Q|P$$
$$P|(Q|P) \equiv (P|Q) \mid R$$
$$(vn.P)|Q \equiv vn.(P|Q) \; when \; n \notin fn(Q)$$
$$vm.vn.P \equiv vn.vm.P$$
$$vn.0 \equiv 0$$

Definition 10.10 (Local Reductions, Stable Processes). *Local reduction step*, written $P \to P'$, represents internal computation between local processes, and is defined as the smallest relation such that

(LComm) $(T).Q|T\sigma \to Q\sigma$

(LRepl) $* (T).Q|T\sigma \to Q\sigma| * (T).Q$

(LMatch) match $T\sigma$ with T in P else $Q \to P\sigma$

(LNoMatch) matgh V with T in P else $Q \to Q$ when $V \neq T\sigma$ for any σ

(LParCtx) $\dfrac{P \to Q}{P|R \to Q|R}$ (LNewCtx) $\dfrac{P \to Q}{vn.P \to vn.Q}$ (LStruct) $\dfrac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$

where σ ranges over substitutions of closed terms for the variables bound in T.

The local process P is *stable* when it has no local reduction step, written $P \not\to$.We write $P \twoheadrightarrow Q$ when $P \to^* \equiv Q$and $Q \not\to$.

System Transitions

We define a labelled transition semantics for configurations, then for systems. Each labelled transition, written $S \xrightarrow{y} S'$, represents a single interaction with the adversary. We let α and β range over input and output labels (respectively from and to the adversary), let γ range over labels, and let φ range over series of labels. We write $S \xrightarrow{\varphi} S'$ for a series of transitions with labels φ. We also rely on structural equivalence for configurations.

Definition 10.11 (Structural Equivalence for Configurations). *Structural equivalence for con-figurations* , written $C \equiv C'$, is defined as the smallest congruence such that:

$C \equiv C'|0 \quad C|C' \equiv C'|C \quad C|(C'|C'') \equiv (C|C')|C''$

$vm.vn.C \equiv vn.vm.C \quad (vn.C)|C' \equiv vn.(C|C') \ when \ n \notin fn(C')$;

Definition 10.12 (Labels). *Labels* are defined by the following grammar:

α::=	input labels
(M)	input of message M
(i)	forwarding of intercepted message i
β::=	output labels
$vn_1 \ldots n_k.M$	output of message M ($n_{1,\ldots,} n_k \in fn(M)$)
$vi.a:b$	nterception of message i from a to b ($a, b \in H$)
γ::=	single label
$\alpha + \beta$	input or output label
φ::=	series of transition labels
γ	

We let *input*(φ) be the series of input labels in φ.

Definition 10.13 (Labelled Transitions for Configurations). *Labelled transitions for configura-tions* are defined by the following rules:

$$(CFGOUT)\ \frac{u \neq a}{a[a:u\langle V\rangle|Q] \xrightarrow{a:u\langle V\rangle} a[Q]}$$

$$(CFGIN)\ \frac{u:a\langle V\rangle|P \to Q \quad u \neq a}{a[P] \xrightarrow{(u:a\langle V\rangle)} a[Q]}$$

$$(CFGBLOCK)\ \frac{C \xrightarrow{b:a\langle V\rangle} C' \quad i \ not \ in \ C}{C \,|\, a[P] \xrightarrow{vi.b:a} C' \,|\, a[P] \mid b:a\langle V\rangle/i}$$

$$(CFGFWD)\ \frac{C \xrightarrow{(M)} C'}{C|\ M/i \xrightarrow{(i)} C'}$$

$$(CFGPRINCTX)\ \frac{C \xrightarrow{\gamma} C' \quad \gamma \ not \ from/to \ a}{C \,|\, a[P] \xrightarrow{\gamma} C' \,|\, a[P]}$$

$$(CFGOPEN)\ \frac{C \xrightarrow{\beta} C' \quad n \ free \ in \ \beta}{vn.C \xrightarrow{vn.\beta} C'}$$

$$(CFGNEWCTX)\ \frac{C \xrightarrow{\gamma} C' \quad n\ not\ in\ \gamma}{\nu n.C \xrightarrow{\gamma} \nu n.C'} \qquad (CFGSTR)\ \frac{C \equiv D \quad D \xrightarrow{\gamma} D' \quad D' \equiv C'}{C \xrightarrow{\gamma} C'}$$

Rules (CFGOUT) and (CFGIN) represent "intended" interactions with the environment, as usual in an asynchronous pi calculus. They enable local processes for any $a \in H$ to send mes-sages to other principals u, and to receive their messages. The transition label conveys the com-plete message content.

Rules (CFGBLOCK) and (CFGFWD) reflect the actions of an active attacker that intercepts messages exchanged between compliant principals, and selectively forwards those messages. In contrast with the (COMM) rule of the pi calculus, they ensure that the environment mediates all communications between principals. The label produced by (CFGBLOCK) signals the message interception; the label conveys partial information on the message content that can intuitively be observed from its wire format: the environment learns that an opaque message is sent by b with intended recipient a. In addition, the whole intercepted message is recorded within the configuration, using a fresh index i. Later on, when the environment performs an input with label (i), Rule (CFGFWD) restores the original message and consumes $M=i$; this ensures that any intercepted message is delivered at most once.

The local-reduction hypothesis in Rule (CFGIN) demands that all local reductions triggered by the received message be immediately performed, leading to some updated stable process Q. Intuitively, this enforces a transactional semantics for local steps, and prevents any observation of their transient internal state. (Otherwise, the environment may for instance observe the order of appearance of outgoing messages.) On the other hand, any outgoing messages are kept within Q; the environment can obtain all of them via rules (CFGOUT) and (CFGBLOCK) at any time, since those outputs commute with any subsequent transitions.

The rest of the rules for configurations are standard closure rules with regards to evaluation contexts and structural rearrangements: Rule (CFGOPEN) is the scope extrusion rule of the pi calculus that opens the scope of a restricted name included in a message sent to the environment; this rule does not apply to intercepted messages. Rule (CFGPRINCTX) deals with principal a defined in the configuration; condition γ not from a excludes inputs from the environment that

would forge a message from *a*, whereas condition γ not to *a* excludes outputs that may be transformed by Rule (CFGBLOCK).

Finally, we have a pair of top level rules that deal with the attacker knowledge:

Definition 10.14 (Labelled Transitions for Systems). *Labelled transitions for systems* are de-fined by the following rules:

$$(SYSOUT)\ \frac{C \xrightarrow{\beta} C'}{\Phi \vdash C \xrightarrow{\beta} \Phi \cup \emptyset(\beta)_H C'} \qquad (SYSIN)\ \frac{C \xrightarrow{a} C' \quad \emptyset(a)_H \subseteq M(\Phi)}{\Phi \vdash C \xrightarrow{a} \Phi \vdash C'}$$

where H is the set of principals defined in C and $M(\Phi) = \{a\{V\}_l : a\{V\}_0 \in \Phi\}$ is the set of certificates the attacker might produce from Φ (see Appendix A for the motivation for this rule).

Rule (SYSOUT) filters every output β and adds to Φ the certificates included in β. Rule (SYSIN) filters every input *a*, and checks that the certificates included in *a* can be produce from the certificates in Φ.

Our main results are expressed using *normal transitions* between systems.

Definition 10.15 (Stable Systems, Normal Transitions). We say that the system *S* is *stable* when all local processes of *S* are stable and *S* has no output transition. (Informally, *S* is waiting for any input from the environment.)

We say that a series of transitions $S \xrightarrow{\varphi} S'$ is *normal* when every input transition is followed by a maximal series of output transitions leading to a stable system, that is, $\varphi = \varphi_1 \varphi_2 \ldots \varphi_n$ where $\varphi_i = a_i \widetilde{\beta_i}$ for $i = 1..n$, and $S = S_0 \xrightarrow{\varphi_1} S_1 \xrightarrow{\varphi_2} S_2 \ldots \xrightarrow{\varphi_n} S_n = S'$ for some stable systems $S_0, \ldots, S_n$.

Intuitively normality states that each principals outputs all his messages and stays idle until he receives a new input.

Compositionality

By design, our semantics is compositional, as its rules are inductively defined on the structure of configurations. For instance, we obtain that interactions with a principal that is implicitly controlled by the environment are at least as expressive as those with any principal explicited within the system.

If we have $C \left| a[P] \xrightarrow{a} C' \right| [P]$, then we also have $C^o \xrightarrow{\beta} C'^0$, where C^o and C'^0 are obtained from *C* and *C'*, respectively, by erasing the state associated with *a*: any intercepted messages *M / i* from *a* or to *a*; and any certificate in Φ issued by *a*. This compositional property yields useful congruence properties for observational equivalence on configurations.

An Abstract Machine for Local Reductions

In preparation to the description of a concrete machine M_a that executes *a*'s local process P_a, we derive a simple algorithm for local reductions. In contrast with our non-deterministic reduction semantics, the algorithm relies on partial normal forms instead of structural equivalence, and it carefully controls the creation of fresh names (to be implemented as random-number generation); it also relies on an explicit scheduler and is otherwise deterministic.

A process *P* is in *normal form for a* when it is a closed well-formed process such that every subprocess of the form match *V* with *T* in *Q* else *Q'* or *vn.Q* appears only under an input or a replicated input—intuitively, all name creations and matchings are guarded. Let *P* be in normal form for *a*. Up to the structural laws for parallel composition, $P \equiv M| \, L \, |G$ where *M* is a parallel composition of messages sent to other principals, *L* is a parallel composition of other (local) messages, and *G* is a parallel composition of inputs and replicated inputs. Concretely, we may represent *P* as a triple of multisets for *M*, *L*, and *G*.

A *scheduler* is a deterministic algorithm on (*L*,*G*) that selects an instance of Rule (LCOMM) or (LREPL) for an output of *L* and an input (or replicated input) of *G*, if any, and otherwise reports completion. The reduction algorithm repeatedly calls the scheduler, performs the selected reduction step, then normalises the resulting process by applying Rules (LMATCH) and (LNOMATCH) and lifting name restrictions to the top level (possibly after a renaming). This yields a local pro-cess of the form $v\tilde{n}$*:*(*M'* | *L'* | *G'*) where $\tilde{n}$ collect all new name restrictions in evaluation context. By induction on the length of the derivation, one easily check that $P \rightarrow Q$ if and only if, for local process is in normal form.

A configuration is in normal form when all restrictions are grouped at top-level and every local process is in normal form. Our local reduction strategy can be extended to configurations in normal forms as follows: we perform local reductions as detailed above, then lift any resulting restrictions to the top level of the configuration up to structural equivalence (using a[v$\tilde{n}$:P']≡ $\nu\tilde{n}$ .a[P'])

10.3 High-Level Equivalences and Safety

Now that we have defined labelled transitions that capture our attacker model and implementation constraints, we can apply standard definitions and proof techniques from concurrency theory to reason about systems. Our computational soundness results are useful (and non-trivial) inasmuch as transitions are simpler and more abstract than low-level adversaries. In addition to trace properties (used, for instance, to express authentication properties as correspondences between transitions), we consider equivalences between systems.

Intuitively, two systems are equivalent when their environment observes the same transitions. Looking at immediate observations, we say that two systems S_1 and S_2 *have the same labels* when, if $S_1 \xrightarrow{\gamma} S_2$ for some S_1' (and the name exported by are not free in S_2), then $S_2 \xrightarrow{\gamma} S_2'$ for some S_2', and vice versa. More generally, bisimilarity demands that this remains the case after matching transitions:

Definition 10.16 (Bisimilarity). The relation R on systems is a labelled simulation when, for all $S_1\ R\ S_2$, if $S_1 \xrightarrow{\gamma} S_1'$ (and the names exported by γ are not free in S_2) then $S_2 \xrightarrow{\gamma} S_2'$ and $S_1'\ R\ S_2'$. Labelled bisimilarity, written ≅, is the largest symmetric labelled simulation.

In particular, if $\Phi \vdash C \cong \Phi' \vdash C'$ then C and C' define the same principals, intercepted-message indices, and exported certificates ($M(\Phi) = M(\Phi')$).

We also easily verify some congruence properties: our equivalence is preserved by addition of principals, deletion of intercepted messages, and deletion of certificates.

Lemma 10.1. *1. If* $\Phi \vdash \mathrm{C}_1 \cong \Phi \vdash \mathrm{C}_2$, *then* $\Phi \cup \Phi_a \; C_1 \mid a[P] \cong \Phi \cup \Phi_a \vdash \mathrm{C}_2 \mid \mathrm{a[P]}$ *for any certificates* Φ_a *issued by a such that the systems are well-formed and* $\emptyset_H(P) \subseteq \Phi$.

If $\Phi \vdash \mathrm{v}\widetilde{\mathrm{n}_1}.(\mathrm{C}_1 \mid \mathrm{M}_1/\mathrm{i}) \cong \Phi \vdash \mathrm{v}\widetilde{\mathrm{n}_2}.(\mathrm{C}_2 \mid \mathrm{M}_2/\mathrm{i})$, *then* $\Phi \vdash \mathrm{v}\widetilde{\mathrm{n}_1}.\mathrm{C}_1 \cong \Phi \vdash \mathrm{v}\widetilde{\mathrm{n}_2}.\mathrm{C}_2$.

If $\Phi \cup \{\mathrm{V}\} \vdash \mathrm{C}_1 \cong \Phi \cup \{\mathrm{V}\} \vdash \mathrm{C}_2$ *and* $V \notin \emptyset(\Phi)$, *then* $\Phi \vdash \mathrm{C}_1 \cong \Phi \vdash \mathrm{C}_2$.

Proof. The proof is by bisimulation. We detail the proof of Property 1 of the lemma—the proofs for the other two properties are similar but simpler. For fixed H ⊂ Prin and $a \in$ Prin \ *H*, we let *R* be the relation defined by: if $\Phi \vdash C_1 \cong \Phi \vdash C_2$, then

$$\Phi_* \vdash v\tilde{n}.(C_1 \mid a[P] \mid C_a) R \; \Phi_* \vdash v\tilde{n}.(C_2 \mid a[P] \mid C_a)$$

for any names $\tilde{n}$, configurations C_1, C_2 that define the principals $b \in H$, local process *P* , parallel composition C_a of intrcepted messages from a or to a? and sets of certificates Φ and Φ_*

such that the systems are well-formed and

$$\emptyset_H(\Phi_*) \cup \emptyset_H(P) \cup \emptyset_H(C_a) \subseteq \Phi$$

We show that *R* is a labelled simulation by case analysis on the transitions of any systems related by *R*, of the form

$$S_1 = \Phi_* v\tilde{n}.(C_1 \mid a[P] \mid C_a) \xrightarrow{\gamma} S_1' = \Phi_* v\tilde{n}'.(C_1' \mid a[P'] \mid C_a')$$

Assuming that S_1 *R* S_2, we establish the existence of a matching transition

$$S_2 = \Phi_* \vdash v\tilde{n}.(C_2 \mid a[P] \mid C_a) \xrightarrow{\gamma} S_2' = \Phi_* \vdash v\tilde{n}'.(C_2' \mid a[P'] \mid C_a')$$

such that S_1' *R* S_2'. We deal with outputs (Rule (SYSOUT)), then inputs (Rule (SYSIN)).

- $\gamma = vi.a{:}\,b$. The transition uses Rule (CFGBLOCK) with index i fresh in $S_1$and $b \in H$ to incept an output produced by Rule (CFGIN):$a[P] \xrightarrow{v\tilde{m}.a:b\langle V\rangle} a[P']$. Up to renaming, we assume that the names $\tilde{m}$ are fresh. The index i is also fresh in S_2.
- To obtain a matching transition with S_1' *R* S_2', we use this P', we let $C_a' = C_a \mid a{:}\,b\langle V\rangle/i$, $\tilde{n}' = \tilde{n}, \tilde{m}$, and we leave the other paremeters unchanged: $C_1' = C_1, C_2' = C_2, \Phi' = \Phi$, and $\Phi_*' = \Phi_*$. Property(3.1) is preserved because $\emptyset(P') \subseteq \emptyset(P)$.

- $\gamma = \nu\tilde{m}.\, a{:}\, e\langle V\rangle$ for some e $\notin H \cup \{a\}$. The transition also uses Rule (CFGIN): we have $a[P] \xrightarrow{\nu\tilde{m}'.a:b\langle V\rangle} a[P']$ for some fresh names $\tilde{m}'$. Let $\tilde{m}'' = fn(V) \cap \tilde{n}$. We have $\tilde{n} = \tilde{n}' \uplus \tilde{m}''$.
- To obtain a matching transition with S_1' R S_2' , we use P' and $\tilde{n}'$, we let $\Phi_*' = \Phi_* \cup \emptyset_{H\cup\{a\}}(V)$ and we leave Φ', C_1 , and C_2 unchanged. Property (3.1) is preserved, as $\emptyset_{H\cup\{a\}}(V) = \emptyset_H(V) \cup \emptyset_{\{a\}}(V)$ and $\emptyset_H(V) \subseteq \emptyset_H(P) \subseteq \Phi$.
- $\gamma = \nu i.\, b : a$ for some $b \in H$. The transition uses Rule (CFGBLOCK) with index i fresh in S_1 and $b \in H$ to intercept an output produced by Rule (CFGOUT): $C_1 \xrightarrow{\nu\tilde{m}b:a\langle V\rangle} C_1'$ for some fresh names $\tilde{m}$.
- By Rule (SYSOUT), we have $\Phi \vdash C_1 \xrightarrow{\nu\tilde{m}b:a\langle V\rangle} \Phi' \vdash C_1'$ where $\Phi' = \Phi \cup \emptyset_{\mathrm{H}}(\mathrm{V})$.
- By bisimilarity hypothesis $\Phi \vdash \mathrm{C}_1 \cong \Phi \vdash \mathrm{C}_2$, we obtain C_1' such that $\Phi \vdash \mathrm{C}_2 \xrightarrow{\nu\tilde{m}b:a\langle V\rangle} \Phi' \vdash \mathrm{C}_2'$ and $\Phi' \vdash \mathrm{C}_1' \cong \Phi' \vdash \mathrm{C}_2'$.
- To obtain a matching transition with $S_1' \, R \, S_2'$, we use C_1', C_2', Φ', we let $C_a' = C_a | b{:}\, a\langle V\rangle$ and $\tilde{n}' = \tilde{n}, \tilde{m}$, and we leave Φ_* and P unchanged.
- $\gamma = \tilde{m}.\, b{:}\, e\langle V\rangle$.
- Similary, the transition uses (CFGOUT): $\Phi \vdash C_1 \xrightarrow{\nu\tilde{m}\prime b:a\langle V\rangle} \Phi' \vdash C_1'$ where $\Phi' = \Phi \cup \emptyset_{\mathrm{H}}(\mathrm{V})$ and, by bisimulation hypothesis, we obtain C_2' such that $\Phi \vdash C_2 \xrightarrow{\nu\tilde{m}b:a\langle M\rangle} \Phi' \vdash C_2'$ with $\Phi' \vdash C_1' \cong \Phi' \vdash C_2'$.
- To obtain a matching transition and S_1' R S_2', we use $C_1', C_2', \Phi', \tilde{n}' = \tilde{n}, \tilde{m}$ and we leave C_a, Φ_*, and P unchanged. Property (3.1) is preserved, since $\emptyset_H(\emptyset_{\{a\}}(M)) \subseteq \Phi$ by hypothesis.
- $\gamma = (e{:}\, u\langle V\rangle)$. We have $\emptyset_{H\cup\{a\}}(V)) \subseteq \Phi_*$ by Rule (SYSIN) and $\emptyset_H(\Phi_*) \subseteq \Phi$ by Property (3.1), so $\emptyset_H(V) \subseteq \Phi$. Up to a renaming of $\tilde{n}$, we assume that the names of V do not clash with $\tilde{n}$. We distinguish two subcases:
- if $u = a$, then $a[P\,] \xrightarrow{\gamma} a[P']$ by Rule (CFGIN).

We use P' and leave the other parameters unchanged. Property (3.1) is preserved: $\emptyset_H(P) \subseteq \Phi$ also by Property (3.1), so $\emptyset_H(P') \subseteq \Phi$ by definition of local reductions and *P* well-formed for *a*.

- otherwise, $u = b$ for some $b \in H$, and $\Phi \vdash C_1 \xrightarrow{\gamma} \Phi \vdash C_1'$ by Rule (CFGIN).

By bisimulation hypothesis, we obtain C_2' such that $\Phi \vdash C_2 \xrightarrow{\gamma} \Phi \vdash C_2'$ and $\Phi \vdash C_1' \cong \Phi \vdash C_2'$

We use C_1' and C_2' , and leave the other parameters unchanged.

$\gamma = (i)$. We similarly conclude in each of the following subcases: M / i to *a*; C_a defines M / i from *a*; or C_1 defines M / i.

Finally, R is symmetric by construction, hence $R \subseteq \cong$, and R contains the systems related by the lemma for $\tilde{n} = \emptyset$, $\Phi = \Phi_*$, and $C_a = 0$.

Bounding processes

As we quantify over all local processes, we must at least bound their computational power. In-deed, our language is expressive enough to code Turing machines and, for instance, one can easily write a local process that receives a high-level encoding of the security parameter ´ (e.g. as a series of η messages) then delays a message output by 2^{η} reduction steps, or even imple-ments an ‘oracle’ that performs some brute-force attacks using high level implementations of cryptographic algorithms.

Similarly, we must restrict non-deterministic behaviours. Process calculi often feature non-determinism as a convenience when writing specifications, to express uncertainty as regards the environment. Sources of non determinism include local scheduling, hidden in the associative-commutative laws for parallel composition, and internal choices. Accordingly, abstract properties and equivalences typically only consider the existence of transitions—not their probability.

Observable non-determinism is problematic in a computational cryptographic setting, as for in-stance a non-deterministic process may be used as an oracle to guess every bit of a key in linear time.

In order to bound the complexity of processes (mainly the complexity of reductions) we de-fine a function $\lceil \bullet \rceil$ that computes the high-level structural size of systems, labels and transitions.

This is done by structural induction, with for instance $\left\lceil S \xrightarrow{\beta} S' \right\rceil = \lceil S \rceil + \lceil \beta \rceil + \lceil S' \rceil + 1$. As for input lables we have that the complexity of $S \xrightarrow{(a)} S'$ accounts also for the internal reductions performed during the transition, that is, $\left\lceil S \xrightarrow{(a)} S' \right\rceil = \lceil S \rceil + \lceil a \rceil + \lceil S' \rceil + \lceil u{:}\, a\langle V \rangle \mid P \rightarrow Q \rceil + 1$,

where $a[P\ ]$ is defined in S and $a[Q]$ is defined in S'. We omit the rest of the details as they are straightforward.

Definition 10.17 (Safe Systems). A system S is *polynomial* when there exists a polynomial p_S

And a constant C such that, for any φ, if $S \xrightarrow{\varphi} S'$ then $\left\lceil S \xrightarrow{\varphi} S' \right\rceil \leq p_s(\lceil input(\varphi) \rceil)$, and $\lceil \beta \rceil \leq c$ for all output labels $\beta$and φ.''

A system S is safe when it is polynimial and φ, if $S \xrightarrow{\varphi} S'$ and $S \xrightarrow{\varphi} S_2$ then S_1 and S_2 have the same lables.

Hence, starting from a safe process, a series of labels fully determines any further observa-tion. Safety is preserved by all transitions, and also uniformly bounds (for example) the number of local reductions, new names, and certificates.

These restrictions are serious, but they are also easily established when writing simple pro-grams and protocols. (Still, it would be interesting to relax them, maybe using a probabilistic process calculus.) Accordingly, our language design prevents trivial sources of non-determinism and divergence (e.g. with pattern matching on values, and replicated inputs instead of full-fledged replication); further, most internal choices can be coded as external choices driven by the inputs of our abstract environment.

We can adapt usual bisimulation proof techniques to establish both equivalences and safety: instead of examining all series of labels φ, it suffices to examine single transitions for the systems in the candidate relation.

Lemma 10.2 (Bisimulation Proof). *Let R be a reflexive labelled bisimulation such that, for all related systems $S_1\ R\ S_2$, if $S_1 \xrightarrow{\gamma} S_1'$ and $S_2 \xrightarrow{\gamma} S_2'$, then $S_1'\ R\ S_2'$.*

Polynomial systems related by R are safe and bisimilar.

Proof. By induction of φ, we show that $S_1 \; R \; S_2$ and $S_i \xrightarrow{\varphi} S_i'$ for $i = 1,2$ implies $S_1' \; R \; S_2'$.

Equivalences with Message Authentication; Strong Secrecy and Authentication

We illustrate our definitions using basic examples of secrecy and authentication stated as equiv-alences between a protocol and its specification (adapted from [19]). Consider a principal *a* that sends a single message. In isolation, we have the equivalence $a[a\colon b\langle V'\rangle]$ if and only if $V = V'$, since the environment observes V on the label of the transition $a[a\colon b\langle V\rangle] \xrightarrow{a:b\langle V\rangle} a[0]$. Consider now the system

$$S(V,W) = a[a\colon b\langle V,W\rangle] \mid b[(a\colon \langle ?\,x, \_\rangle).P],$$

with an explicit process for principal *b* that receives *a*'s message and, assuming the message is a pair, runs *P* with the first element of the pair substituted for *x*. For any terms W_1 and W_2, we have $S(V; W_1) \cong S(V; W_2)$. This equivalence states the strong secrecy of W , since its value cannot affect the environment. The system has two transitions

$$S(V,W) \xrightarrow{vi.a:b\ (i)} \to a[o] \mid b[P\{V/x\}]$$

interleaved with inputs from any $e \in$ Prin $\setminus \{a,b\}$. Further, the equivalence

$$S(V,W) \cong a[a\colon b\langle\ \ \rangle] \mid b[(a\colon \langle\_\rangle).P\left\{\frac{V}{x}\right\}]$$

captures both the authentication of *V* and the absence of observable information on *V* and *W* in the communicated message, since the protocol *S*(*V*, *W*) behaves just like another protocol that sends a dummy message instead of *V*, *W* .

Equivalences with Certificates

Let Φ ={a{m}} —that is, assume *a* has issued a single certificate. We have

$$\Phi \vdash a[(e\colon \langle a\{n\}\rangle).P] \cong \Phi \vdash a[]$$

$$\Phi \vdash a[a\colon b\langle a\{n\}\rangle \mid (e\colon \langle a\{n\}\rangle).P] \mid b[] \cong \Phi \vdash a[a\colon b\langle 0\rangle] \mid b[]$$

$$\Phi \vdash a[(e\colon \langle a\{x\}\rangle).P] \cong \Phi \vdash a\{(e\colon \langle a\{\_\}\rangle).P\left\{\frac{m}{x}\right\}]$$

These three equations rely on the impossibility for the adversary to forge any certificate from *a* with another content. Similar equations also hold if the input is performed by another principal (as long as *a* does not issue any other certificate), and even if the attacker can choose arbitrary values *V* and *W* instead of the names *m* and *n*, as long as $V \neq W$. Conversely, consider the system

$S[\_]$

$= \Phi \vdash a[(e{:}\langle a\{m\} as\ sig, a\{m\}\ as\ sig'\rangle).\ match\ sig\ with\ sig'\ in\ 0\ else\ [\_]]$

Since signatures are malleable, the else branch is reachable. Take as an example, an input labelled $9e{:}\ a\langle a\{m\}_0, a\ \{m\}_1\rangle$, hence in general $S[P] \not\cong S[Q]$.

10.4 Applications

We present three coding examples within our language, dealing with anonymous forwarders, electronic contracts, and system initialisation. In addition, we coded a translation from asyn-chronous pi calculus processes into local processes, using terms chan(*n*) to represent channels. (The scope of name *n* represents the scope of the channel, and channel-based communications is implemented by pattern matching on channel terms.) We also coded distributed communica-tions for the authenticated join-calculus channels of [13], using certificates $a\{\text{chan}(n)\}$ to represent output capabilities of channels.

Anonymous Forwarders

We consider a (simplified, synchronous) anonymising mix hosted by principal *c*. This principal receives a single message *V* from every participant $a \in A$, then forwards all those messages to some sender-designated address *b*. The forwarded message does not echo the sender identity— however this identity may be included as a certificate in the message *V*. We study a single round, and assume that, for this round, the participants trust *c* but do not trust one another. We use the following local processes (indexed by principal) and systems:

$$P_c = \Pi_{a\in A}(a{:}\ c\langle ?\,b, ?\,V\rangle).\,(tick|(go).\,c{:}\ b\langle forward(V)\rangle)$$

$$Q_c = (tick).^{for\ each\ a\in A}\ \Pi_{a\in A}\ go$$

$$P_a^{\sigma} = a{:}\ c\langle b_{a\sigma}, V_{a\sigma}\rangle \mid P_a'$$

$$S^{\sigma} = c[P_c \mid Q_c]\ \mid \Pi_{a\in A'}\ a[P_a^{\sigma}]$$

The process P_c receives a single message from every $a \in A$, then it emits a local tick message and wait for a local go message. The process Q_c runs in parallel with P_c and provides synchro-nisation; it waits for a tick message for every participant, then sends go messages to trigger the forwarding of all messages.

Let $A' \subseteq A$ be a subset of participants that comply with the protocol. We set $H = A' \uplus \{c\}$. Anonymity for this round may be stated as follows: no coalition of principals in $A \setminus A'$ should be able to distinguish between two systems that differ only by a permutation of the messages sent by the participants in A'. Formally, for any such permutations σ and σ', we verify the equivalence $S^{\sigma} \cong A^{\sigma'}$. Hence, even if the environment knows all the V messages, the attacker gains no information on σ. (Conversely, the equivalence fails, due to traffic analysis, if we use instead a naive mix that does not wait for all messages before forwarding, or that accepts messages from any sender.)

Electronic Payment Protocol

As a benchmark for our framework, we consider the electronic payment protocol presented by Backes and Durmuth¨ [14] that is a simplified version of the 3KP payment system [15, 16]. We refer to their work for a detailed presentation of the protocol and its proper-ties. The authors provide a computationally sound implementation of the protocol on top of an idealised cryptographic library [17]. We obtain essentially the same security properties, but our coding of the protocol is more abstract and shorter than theirs (by a factor of 10) and yields simpler proofs, essentially because it does not have to deal with the details of signatures, marshalling, and local state—coded once and for all as part of our language implementation.

We adapt their notations, e.g. $d, p \mapsto t$. Our calculus is more abstract and formally conve-nient, but less expressive than machines running on top of their library. Arguably, our low-level machine description factors out (and clarifies) most of their coding on top of the library.

The protocol has four roles, a client c, a vendor v, an acquirer ac, and a trusted third party ttp. For simplicity, we assume that ac and ttp are unique and well-known. In addition, we use a distinct, abstract principal U that sends or receives all events considered in trace

properties. Initially, the client, vendor, and acquirer tentatively agree on their respective identities and a (unique) transaction descriptor t that describes the goods and their price. The protocol essentially relies on the forwarding of certificates. We let $x{:}\{y, V\}$ abbreviates a message with a certified content $x{:}\,y\langle x\{y, V\}\rangle$, and use *as sig* to bind the corresponding certificate $x\{y, V\}$.

A system S consists of any number of principals (potentially) running the three roles, plus a unique principal ttp running P_{ttp}. The system should not define U, which represents an arbitrary, abstract environment that controls the actions of the other principals. For a given normal trace φ, we say that the payment t, c, v, ac is *complete* when φ includes the following input labels:

- if c ∈ H, then $U{:}\,c\langle pay(t, v)\rangle$;
- if v ∈ H, then $U{:}\,v\langle receive(t, c)\rangle$; and
- if ac ∈ H, then $U{:}\,ac\langle allow(t, c, v)\rangle$.

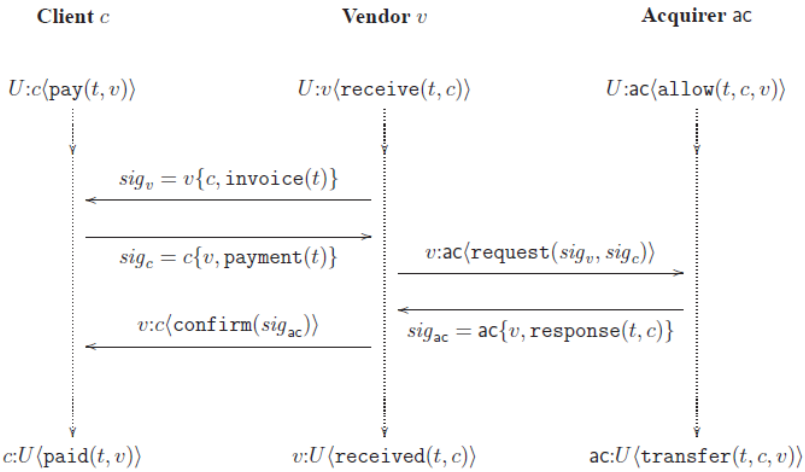


Figure 10.1: Diagram of the Electronic Payment Protocol [14]

We can now state the following properties:

- Weak atomicity is a trace property expressed as follows: if φ includes any output of the form $c{:}\,U\langle paid(t, v)\rangle, v{:}\,U\langle received(t, c)\rangle$, or

$ac: U\langle transfer(t,c,v)\rangle$, then the payment t, c, v, ac is complete.

- Correct client dispute states that an honest client—who starts a dispute for transaction t only after completing the protocol for t, as coded in the last line of Clientc—always
- wins his dispute: that is, for any trace φ, if c ∈ H and $c: U\langle paid(t,v)\rangle$ is in φ, then $ttp: U\langle reject(t,c,v)\rangle$ is not in φ. (This property is rather weak, as the vendor and acquirer complete the protocol before the client.)
- Correct vendor dispute and Correct acquirer dispute are similar to the previous property and we omit it here.
- No framing states that the ttp does not wrongly involve parties that have not initiated the protocol with matching parameters. It is a variant of weak atomicity: outputs of the form $ttp: U\ \langle accept(t,c,v)\rangle$ only occur for complete payments.

These properties are directly established by induction on the high-level transitions of *S*.

Sketch of the Proof. By induction on the trace φ. We show that the state of the system is de-termined by φ, and that every enabled input in this state yields outputs that meet the claimed properties. (In contrast with [14], we don't have to define complex, auxiliary invariants; the invariant directly follows from our definition of labelled transitions.)

$Client_c$ = $*(U{:}c\langle \text{pay}(?\,t, ?\,v)\rangle).$
$(v:\{c,\text{invoice}(t)\}\ as\ sig_v).$
$(c:\{v,\text{payment}(t)\}\ |$
$(v{:}c\langle \text{confirm}(\text{ac}\{v,\text{response}(t,c)\}\ as\ sig_{\text{ac}})\rangle).\ ):$
$(c{:}\text{U}\langle \text{paid}(t,v)\rangle\ |$
$(U: c\langle dispute(t)\rangle).\text{c:ttp}\langle client\_dispute(sig_v, sig_{ac})\rangle))$

$Vendor_v$ = $*(U{:}v\langle \text{receive}(?\,t, ?\,c)\rangle).$
$(v:\{c,\text{invoice}(t)\}\ as\ sig_v\ |$
$(c:\{v;\text{payment}(t)\}\ as\ sig_c).$
$(v{:}\text{ac}\langle request(sig_v,\ sig_c)\rangle\ |$
$(\text{ac}:\{v,\text{response}(t,c)\}\ as\ sig_{\text{ac}}).$
$(v{:}c\langle confirm(sig_{ac})\rangle\ |\ v{:}U\langle \text{received}(t,c)\rangle\ |$
$(U: v\langle dispute(t)\rangle).\ v: tpp\langle vendor\_dispute(sig_c, sig_{ac})\rangle)))$

$Acquirer_{ac}$ = $*\ (U: ac\langle allow(?\,t, ?\,c, ?\,v)\rangle).$

$$
\begin{aligned}
&(v{:}\, ac\langle request(v\{c, invoice(t)\}\ as\ sig_v,\\
&\qquad c\{v, payment(t)\}\ as\ sig_c)\rangle).\\
&(ac{:}\,\{v, response(t,c)\} \mid ac{:}\, U\langle transfer(t,c,v)\rangle \mid\\
&(U{:}\, ac\langle dispute(t)\rangle).\, ac{:}\, ttp\langle acquirer\_dispute(sig_c, sig_v)\rangle)\\
&\qquad P_{ttp} = *\,(?\,c{:}\, ttp\langle client\_dispute(?\,d)\rangle).\\
&\quad match\ d\ with\ ?\,v\{c, invoice(?\,t)\}, ac\{v, response(t,c)\}\ in\\
&\qquad ttp{:}\, U\langle accept\_client(t,c,v)\rangle\ \ else\\
&\qquad ttp{:}\, U\langle reject\_client(t,c,v)\rangle \mid\\
&\qquad *\,(?\,v{:}\, ttp\langle vendor\_dispute(?\,d)\rangle).\\
&\quad match\ d\ with\ ?\,c\{v, payment\ (?\,t)\}, ac\{v, response(t,c)\}\ in\\
&\qquad ttp{:}\, U\langle accept\_vendor(t,c,v)\rangle\ \ else\\
&\qquad tpp{:}\, U\langle reject\_vendor(t,c,v)\rangle \mid\\
&\qquad *\,(ac{:}\, tpp\langle acquirer\_dispute(?\,d)\rangle).\\
&\quad match\ d\ with\ ?\,c\{?\,v, payment(?\,t)\}, v\{c, invoice(t)\}\ in\\
&\qquad ttp{:}\, U\langle accept\_acquirer(t,c,v)\rangle\ \ else\\
&\qquad ttp{:}\, U\langle reject\_acquirer(t,c,v)\rangle
\end{aligned}
$$

Figure 10.2: Encoding of the Electronic Payment Protocol [14]

Initialisation

This technical example shows that, without loss of generality, it suffices to develop concrete implementations for *initial systems* that do not share any names, certificates, or intercepted mes-sages between principals and the environment. Up to structural equivalence, every system is of the form $S = \Phi \vdash \nu\tilde{n}.\,(\Pi_{a\in H} a[P_a] \mid\mid \Pi_{i\in I}\, M/i$. The sharing of names and certificates between principals and the environment can be quite complex, and is best handled using an ad hoc (but high-level) "bootstrapping" protocol, outlined below:

- Free names of *S* and restricted non-local names *n*e are partitioned between honest princi-pals; let $(n_{a,1}, \ldots, n_{a,k_a})_{a\in H}$ be those names.
- Free names and non-self-issued certificates that occur in the local processes P_a are ex-changed using a series of initialisation messages $M_{ab,r}$ of the form

$$M_{ab,r} = a{:}\, b\,\langle init_{ab,r}(n_{a,1}, \ldots, n_{a,k_{a_r}}, a\{V_{ab,1}\}, \ldots, a\{V_{ab,m_r}\})\rangle,$$

carrying names and certificates issued by a that occur in P_b. Similarly, initialization mes-sages sent to a fixed principal $e \notin H$ export

the free names of S and the certificates of Φ, whereas initialization messages from e import certificates issued by principals not in H.

Each principal $a \in H$ thus sends a series of initialisation messages, and sequentially re-ceives and checks all initialisation messages addressed to him, using input patterns of the form $(T_{ba,r})$ where $T_{ba,r}$ is $M_{ba,r}$ with binding variables $?n_1, \ldots, ?n_k$ instead of the names and aliases $b\{V_{ba,r}\}$ *as* $?x$ for checking and binding certificates. The whole local initiali-sation process is guarded by a dummy input with pattern $T_{ea,0} = e{:}\, a\langle\_\rangle$, so that the initial system be stable.

- Finally, each principal a sends a message M for every intercepted message M/i from a defined in S, then starts P_a.

For instance, in case $H = \{a, b\}$ with neither nested certificates nor intercepted messages, the local initialisation process for a is

$$P_a^o = (T_{ea,0}).\, \nu n_1, \ldots, n_{k_a}.\, (M_{ab,1} \mid M_{ae,1} \mid (T_{ba,1}).\, (T_{ea,1}).\, P_a)$$

In the general case, several rounds of initialisation messages may be needed to exchange certifi-cates whose contents include names and certificates, and to emit messages with the same shape one at a time.

Intuitively, the attacker may prevent P_a from running at all by not forwarding messages, or provide a message whose certificates do not match the certificates expected by P_a, but it could block all of a's communications anyway. If P_a does start, it does so with the right names and certificates.

The next lemma states the correctness of the initialisation protocol. The second property of the lemma states that an environment that follows the protocol always reaches S_i.

Lemma 10.3 (Initialisation). *Let S_i for $i = 0, 1$ be safe stable systems with the same principals, exported certificates, and intercepted-message labels.*

There exist initial safe stable systems S_i^o and labels φ^o such that

- we have normal transitions $S_i^o \xrightarrow{\varphi^o} S_i$;
- any normal transitions $S_i^o \xrightarrow{\varphi^o} S'$ imply that $S' \equiv S_i$; and
- $S_0 \cong S_1$ if and only if $S_0^o \cong S_1^o$.

Proof. (Sketch.) We have $S_i^o \xrightarrow{\varphi^o} S_i$ deterministically, so $S_0^o \cong S_1^o$ implies $S_0 \cong S_1$. Conversely, we show that the relation

$$R = \{S_0', S_1'\} \text{ such that } S_0 \cong S_1, S_i^o \xrightarrow{\varphi^o} S_i,$$

and φ is a prwfix of a permutation of the lables of $\varphi^o\} \cup \cong$

is a labelled bisimulation. (Intuitively, φ is the part of φ^o that has already been enabled by the attacker.)

10.5 A Concrete Implementation

We are now ready to define the machines, relying on translations from high-level terms and processes to keep track of their runtime state. We systematically map high-level systems *S* to the machines, mapping each principal $a[P_a]$ of *S* to a PPT machine M_a that executes P_a. We start by giving an outline of our implementation.

The implementation mechanisms are simple, but they need to be carefully specified and com-posed. (As a non-trivial example, when a machine outputs several messages, possibly to the same principals, we must sort the messages after encryption so that their ordering on the wire leaks no information on the computation that produced them.)

We use two concrete representations for terms: a wire format for (signed, encrypted) mes-sages between principals, and an internal representation for local terms. Various bitstrings rep-resent constructors, principal identities, identifiers for names, and certificates. Marshaling and unmarshaling functions convert between internal and wire representations. When marshaling a locally restricted name identifier *ind* for the first time, we draw a bitstring *s* of length ´ uniformly at random, associate it with *ind*, and use it to represent *ind* on the wire. When unmarshaling a bitstring *s* into an identifier for a name, if *s* is not associated with any local identifier, we create a new internal identifier *ind* for the name, and also associate *s* with *ind*.

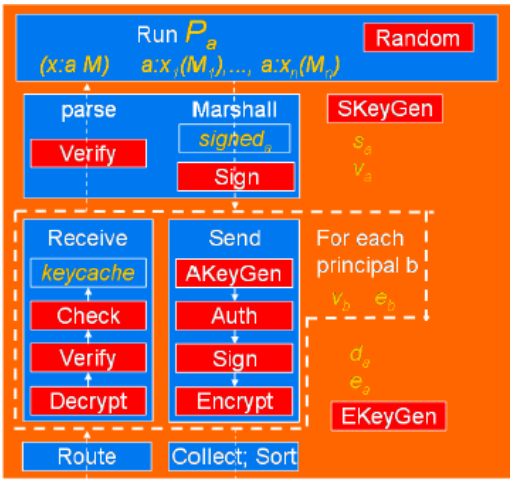


Figure 10.3: Local machine for principal *a* connected to the adversary machine

Signatures are verified as part of unmarshaling. Signatures for self-issued certificates are generated on-demand, as part of marshaling, and cached, so that the same signature value is used for any certificate with identical content.

Local processes are represented in normal form for structural equivalence, using internal terms and multisets of local inputs, local outputs, and outgoing messages. We implement reduc-tions using an abstract machine that matches inputs and outputs using an arbitrary deterministic, polynomial-time scheduler.

Implementation of Machines

The transition rules declare that all communications be authentic and confiden-tial. In order to meet these requirements, our implementation relies on concrete bitstrings and cryptographic protocols.

Definition 3.18 (Low-Level State). The runtime state of machine M_a consists of the following data:

- id_a, d_a, and s_a are bitstrings that represent the low-level identifier for principal *a* and its private keys for decryption and signing.

- *peers* = {(id_u, e_u, v_u) | $u \in$ Prin} binds, for every principal, a low-level identifier to public keys for encryption and signature verification.
- p_a is a low-level representation of a local process running at *a* (defined below).
- $keycache_a$ is a set of authentication keys for all received messages.
- $signed_a$ is a partial function from certificates issued by *a* to signature values.
- $names_a$ is a partial function from name identifiers to bitstrings.

The main machine components are depicted in Figure 10.3.

Before detailing the definitions of all the protocols presented if Figure 10.3, we describe a complete run of the machine. Recall that M_a is connected to the environment by two wires, in_a and out_a. The wire format for messages is the concatenated bitstring $id_u\_id_v\_msg$ where *u* and *v* are the (apparent) sender and receivers and *msg* is some encrypted, authenticated, marshaled message. When it receives such a message (with $id_v = id_a$), M_a uses id_u to dispatch *msg* to the receive protocol (Definition 3.23) for remote principal *u*— there is an instance of the receive protocol for each peer principal *u*. The protocol verifies the freshness, integrity, and authenticity of the message, updates $keycache_a$, then returns a decrypted bitstring *s*. If a verification step fails, the message is discarded.

At this stage, *msg* is a genuine message from *u* to *a*, but its content is not necessarily well-formed. For instance *u* may have included a certificate apparently issued by *b* but with an invalid signature. Content validation occurs as *s* is unmarshaled (Definition 10.21) from its wire format into some internal (trusted) representation $parse_a(s)$ of a high-level term *V*. In particular, this trusted representation embeds a valid signature for every certificate of *V*. After successful recep-tion and unmarshaling, a representation *m* of the incoming message $u{:}a\langle V\rangle$ *i*may react with an input within p_a and trigger local computations. To this end, a local interpreter (Definition 10.19) derived from the abstract machine runs on $p_a \mid m$. If the interpreter terminates, it yields a new stable internal process p'_a plus a set of outgoing messages *X* to be sent to the network.

Each message $a{:}u_i\langle V_i\rangle$ represented in *X* is then marshaled (Definition 10.20) and passed to the instance of the send protocol

(Definition 10.22) associated with the intended recipient u_i. The resulting bitstrings, all in wire format, of the form $id_a\_id_{ui}\_msg_i$, are eventually sorted (by receiving principal, then encrypted value msg_i)—to ensure that their ordering leaks no informa-tion on their payload or their internal production process—and written on out_a. A final done bitstring is issued and the machine terminates. (Hence, for instance, if p does not react with m, the machine simply writes done on out_a and terminates.)

Next, we describe in turn each of the components of the local machine.

Low-level Processes Reductions

The internal representation of terms uses the same grammar as in the high-level language except for atomic subterms: principals u are boxed, fixed-sized bitstrings prin(id_u) (l_{prin}); free names are boxed, bitstrings name(ind) where ind is an internal identifier for names; and certificate labels are linear-sized bitstrings s such that either s is a valid signature for the certificate or $s = 0$ and the certificate is self-issued. Bound variables and names may still occur in terms under input guards.

Definition 10.19 (Internal Reductions). The local reduction algorithm refines the abstract machine as follows:

- it represents the multisets X, M, and G using internal terms;
- it uses a deterministic, polynomial-time, complete scheduler;
- instead of lifting new name restriction $vn.Q$, it generates a new identifier ind (possibly incrementing an internal counter) and substitutes name(ind) for all bound instances of the n in Q.

Marshaling and Unmarshaling Protocols

These algorithms are responsible for processing messages that are about to be sent to (that were received from) the network. The marshaling process transforms each internal term into a bit-string to be sent over the network, and the unmarshal algorithm attempts to transform a bitstring received from the network to a (trusted) internal term; it may instead return an error if the mes-sage is not well-formed, or if the signature of an included certificate cannot be verified. In any of these cases the entire message is discarded.

We use a fixed, injective function from all constructors plus name and prin to bitstrings of a given fixed size; we still write f, name, prin for the corresponding bitstrings. We write s_s' for the bitstring obtained by concatenating s and s'.

Definition 10.20 (Marshaling). Let $\Sigma = (G, S, V)$ be a signature scheme. The function $[\![\bullet]\!]$ maps principal's internal representations of closed terms to bitstrings, as follows:

$$[\![name\ (ind)]\!] = name\_names(ind)$$

$adding\ names(ind) = s \leftarrow \{0,1\}^{\eta}$ when undefined

$$[\![prin(s)]\!] = prin\_s$$

$[\![f(v_1, \dots, v_n)]\!] = f\_[\![v_1]\!]\_ \dots \_ [\![v_n]\!]$ when f ∉ {name, prin, cert}

$[\![v_1\{v_2\}_s]\!] = cert\_[\![v_1]\!]\_[\![v_2]\!]\_$ s when $s \neq 0$

$[\![v_1\{v_2\}_0]\!] = cert\_[\![v_1]\!]\_[\![v_2]\!]\_$ signed $(v_1\{v_2\}_0)$ when $v_1 = prin(id_b)$, $b \in H$

adding signed $(v_1\{v_2\}_0) = S(s_b, [\![v_2]\!]_b)$ when undefined.

We denote by $[\![\bullet]\!]_a$ the marshaling procedure for machine M_a that uses only uses tables $names_a$ and $signed_a$, that is, $names = names_a$ and $signed = signed_a$.

We prove as an invariant that for all certificates of the form $v_1\{v_2\}_0$ in p_a, v_1 = prin(id_a), hence $[\![\bullet]\!]_a$ is defined for all internal representations of terms of M_a.

We assume that after marshalling, and before sending, all our messages are padded to a fixed length that is given by the polynomial $ms(\eta)$, a parameter of the implementation. We could have assumed that this was not the case and if so, we needed to consider this difference of length in our high-level semantics. We could have done it using sorts and sizes for input and output messages.

Definition 10.21 (Unmarshaling). Let $\Sigma = (G, S, V)$ be a signature scheme. The partial function $parse(\bullet)$ maps bitstrings to internal representations of closed terms, as follows, and fails in all other cases.

parse(name_s) = name(ind) when, there is ind:names(ind) = s

otherwise if |s| = η then

ind = |dom(names)| + 1 and

add names(ind) = s

parse(prin_s) = prin(s) when $|s| = l_{prin}$ and $(s, e_s, v_s) \in$ peers

parse(F_s_1_... s_n) = $f(v_1, \dots, v_n)$ when f ∉ {name, prin, cert}

has arity n

$\text{parse}(s_i) = v_i$ for $i = 1..n$

$\text{parse}(\text{cert\_}s_1\text{\_}s_2\text{\_}s_3) = v_1\{v_2\}_s$ when, for some $(id_u, e_u, u_u) \in$ peers,

$\text{parse}(s_1) = \text{prin}(id_u) = v_1$,

$\text{parse}(s_2) = v_2$

$\text{V}(u_u, s_2, s_3) = 1$

$\text{s} = \text{if signed}(v_1\{v_2\}_0) = s_3$ then else s_3

We denote by $parse_a(\bullet)$ the unmarshaling procedure for machine M_a that uses only uses tables $names_a$ and $signed_a$, that is, $names = names_a$ and $signed = signed_a$.

Unmarshaling includes signature verification for any received certificate, and is otherwise standard; it is specified here as a partial function from strings to internal representations, and can easily be implemented as a parser. Our treatment of self-issued certificates with label 0 reflects our choice of internal representations: 0 stands for the (unique) signature generated by the local machine for this certificate content, the first time this certificate is marshaled. (In addition, the adversary may be able to derive a variant of this certificate with a different signature, unmarshaled with a non-zero label; such certificates are then treated using the default case for marshaling.)

Although we give a concrete definition of $[\![\bullet]\!]$, $parse(\bullet)$, and message formats, our results only depend on their generic properties. We only require that, for a given local machine, every string be unmarshaled to at most one internal term, whose marshaling yields back the original string, that is, $parse_a\ ([\![\ulcorner V \urcorner^a]\!]_a) = \ulcorner V \urcorner^a$. ($\ulcorner V \urcorner^a$ denotes the internal representation of a for V. We define $\ulcorner V \urcorner^a$ formally in Definition 10.25.) For simplicity, we have that the length of the string be a function of the structure of the internal term and of the security parameter.

Sending and Receiving Protocols

Two important pieces of our systems are the $send_b$ and $receive_b$ protocols. There are one pair of these for each other principal. The send protocol defined below, ensures that, as abstracted in the high-level semantics, all communications are opaque for the adversary using

public-key encryption, and that the communication is authentic, using authentication and signature schemes. This protocol takes a bitstring *s* (containing a marshaled message from *a* to *u*), protects it, and returns it in wire format. Conversely, the receiving protocol takes a message in wire format presumably from *u*, verifies it, and returns its payload. We also request robustness against replay attacks; after decryption, we reject any message whose authentication key is already recorded.

These protocols are intended as a simple example; other choices are possible. We may for instance consider long term shared keys between principals, in order to reduce the overhead of public-key cryptography. If we decide to do so, we should introduce a nonce in the message that is encrypted in Step 3.

Definition 10.22 (Sending to *u*). Let $\Pi = (K, E, D)$, $\sum = (G, S, V)$, and $\Delta = (G_\Delta, A, C)$ be respectively an encryption, signature, and authentication schemes. Given a bitstring *s*, the send$_u$ protocol

- generates a fresh authentication key $k \leftarrow G_\Delta(1^\eta)$;
- computes $m = s\_id_a\_k\_S(s_a, k_{id_u})\_A(k, s)$;
- computes $msg = E(e_u, m)$; and
- retur $id_a \_id_u \_msg$.

Definition 10.23 (Receiving from *u*). Let $\Pi = (K, E, D)$, $\sum = (G, S, V)$, and $\Delta = (G_\Delta, A, C)$ be respectively an encryption, signature, and authentication schemes. Given a bitstring $id_u\ id_a\ msg$, the receive$_u$ protocol

- computes $s\_ id_u \_k \_s_{sig} \_s_{auth} = D(d_a, msg)$;
- checks that there is an entry $(id_u,\ e_u,\ v_u) \in peers$ with $V(v_u, k\_id_a, s_{sig}) = 1$;
- checks that $C(k, s, s_{auth}) = 1$;
- checks that *k* is not in *keycache*, and adds it to *keycache*;
- returns *s*.

The entire message is discarded if any step of the protocol fails.

Mapping High-Level Systems to Low-Level Machines

In order to systematically relate the runtime state of low-level machines to the abstract state of high-level systems, we define an associated *shadow state*. This structure provides a consistent interpretation of terms across machines. In combination, a system and its shadow state deter-mine their implementation, obtained as a

compositional translation of terms, local processes, and configurations. (This state is shadow as it need not be maintained at runtime in the low-level implementation; it is used solely as an abstraction to reason about the correctness of our imple-mentations.) We further partition this state into public parts, intended to be part of the attacker's knowledge, and private parts.

Definition 10.24 (Shadow State). Let $S = \Phi \vdash \nu\tilde{n}.C$ be a system such that the configuration $C = \Pi_{a\in H} a[P_a] \mid\mid \Pi_{i\in I} M/i$ is in normal form. A *shadow state* for *S*, written D, consists of the following data structure:

- prin ∈ Prin → $(\{0,1\}^\eta)^5$ is a function from u ∈ Prin to bitstrings idu, eu, vu, du, su such that u → idu is injective, and for every u ∈ H, we have (eu, du) ←K(1^η), and (vu, su) ← G(1^η).
- The bitstrings idu , eu , vu are public for all u ∈ Prin; du and su are public if u ∈ Prinn\H.
- name ∈ Name → $\{0,1\}^\eta$ is a partial injective function defined at least on every name that occurs free in S, and names that occur in Φ, D.certval or D.wire.
- The bitstring name(m) is public for every name m $\notin \tilde{n}$.
- ni is a family of partial injective functions ni a : Name → $\{0,1\}^\eta$ for each a ∈ H, defined at least for all names of Pa that are not locally-restricted.
- certval is a partial function from certificates $u\{V\}_l$ to s ∈ $\{0,1\}^\eta$ defined at least on the certificates of Φ, D.wire, and all certificates of P_a of the form $a\{V\}_l$ with $l \neq 0$ or $u\{V\}_l$ with $u \neq a$. It is also defined for all the certificates in V such that $u\{V\}_l$ is defined in certval, certval satisfies the following property: if certval ($u\{V\}_l$) = s, then
- The bitstrings s and del are public.
- $\mathcal{V}(v_u, [\![\ulcorner V \urcorner^{\mathsf{D},u}]\!], s) = 1.$
- wire is a partial function from indices i to (M, k, s, del) defined at least on I, where $M = a{:}\,b\langle V\rangle$ with a, b ∈ H, and del = 0 if i ∈ I and del = 1 otherwise. The bitstrings s and k are the output and the authentication key produced by $send_b$ on input $[\![\ulcorner V \urcorner^{\mathsf{D},a}]\!]$.

- keycache is a function from $a \in H$ to sets of bitstrings such that, if exists an i with wire(i) = (M, k, _, 1) with M to a, then k ∈ keycache(a).
- $ms^D(\eta)$ is a polynomial that sets the padding-size of the implementations of S.

Intuitively, *wire* records all messages sent between honest principals; *keycache*(*a*) records the authentication keys of all messages received by *a* so far; it contains at least the keys of mes-sages in *wire* that were already received by *a*. When D is clear from the context, we write *prin*(*a*) instead of D.*prin*(*a*), and similarly for the other components of D. We denote by *public*(D) the binary representation of the public parts of D. When we are not interested in the specific bit-strings, we call it *shape of* D.

Definition 10.25 (Concrete Terms and Processes). A shadow state D and a set of principals

$X \subseteq$ Prin, define a partial map from high-level terms V to internal terms as follows:

- $\ulcorner n \urcorner^{\mathrm{D},X} = \begin{cases} \mathtt{name}(ind) & \text{, if } ind = ni^u(n) \text{ for all } u \in X \\ \bot & \text{, otherwise} \end{cases}$
- $\ulcorner u \urcorner^{\mathrm{D},X} = \mathtt{prin}(\pi_1(prin(u)))$ for any principal $u \in$ Prin;
- $\ulcorner u\{V\}_0 \urcorner^{\mathrm{D},X} = \ulcorner u \urcorner^{\mathrm{D},X} \{\ulcorner V \urcorner^{\mathrm{D},X}\}_0$, if $u \in X$;

$$n^{D,X} = \{fgb$$

- $\ulcorner u\{V\}_\ell \urcorner^{\mathrm{D},X} = \ulcorner u \urcorner^{\mathrm{D},X} \{\ulcorner V \urcorner^{\mathrm{D},X}\}_s$ where $s = certval(u\{V\}_\ell)$ $(u \in$ Prin$)$;
- $\ulcorner \mathtt{f}(V_1, \ldots, V_n) \urcorner^{\mathrm{D},X} = \mathtt{f}(\ulcorner V_1 \urcorner^{\mathrm{D},X}, \ldots, \ulcorner V_n \urcorner^{\mathrm{D},X})$ for any $f \neq \mathtt{cert}$ with arity n.

We extend this map to translate local processes to low-level processes, as follows: high-level terms within local processes are translated as above, except for variables and locally-restricted names (left unchanged); high-level patterns are translated by applying the translation to all high-level terms in the pattern and leaving the rest unchanged; local processes *P* are translated to internal processes $P^{\mathrm{D},X}$ by translating their high-level terms to internal terms.

As a corollary, we have that if D is a shadow state for *S*, and $a \in$ Prin then $\bullet^{\mathrm{D},a}$ is defined for every subterm and subprocess of *S* and D (•

^{D,a} denotes • $^{D,\ fag}$). We often write V instead of $V^{D,a}$ when D and a are clear from the context. Our intent is that, with overwhelming probability, we have $V = V'$ iff $V^{D,a} = V'^{D,a}$ whenever D defines these representations.

We would like to point out that the previous definition is well-formed. One should first notice that we do not translate high-level terms (hence, high-level certificates) with variables and locally-restricted names. Hence, when applying $u\{V\}_l^{D,a}$, we can be sure that the certificate was previously generated and hence defined in D:*certval*.

Definition 10.26(System Implentations). Let S be a system with shadow state D. The implementation of S and D is the collection of machines M(S,D) = $(M_a(S,D))_{a\in H}$ where each machine $M_a(S,D)$ has the following state:

- id_a , d_a , s_a , $peers_a$ are read from D:prin;
- $p_a = P_a^{D,a}$;
- $keycache_a$ = keycache(a);
- $signed_a(\ a\{V\ \}_0^{D,a}) = certval(a\{V\}_0)$ when defined;
- $names_a(ni^a(n)) = name(n)$ when defined,

and uses $[\![\bullet]\!]_a$ and $parse_a(\bullet)$ as the marshaling and unmarshaling algorithms, and $ms^D(\bullet)$ as the padding size.

10.11 Main Results

In this section we present the main results of this Chapter. Throughout this section we assume that the encryption scheme Π = (*K, E, D*) is CCA-2 secure, and the signature scheme Σ = (*G, S, V*) and authentication scheme Δ = (G_Δ, *A, C*) are CMA-secure.

Our main theorems are stated in terms of arbitrary systems *S*. As it is convenient to have a for-mulation of these theorems in terms of arbitrary systems, one should not forget that an arbitrary system *S* is obtained starting from an initial system S^o that has no shared names or certificates and no intercepted messages so, whenever we refer to a system *S*, we are in fact referring to its initial state S^o plus its initialisation procedure. The same happens with the implementations and for that we introduce the notion of *valid shadow*. Intuitively, a shadow D is a valid shadow for *S*, if there is an interactive run (Definition 3.1) that starts with M(S^o, D^o) and leads the machine to state M(*S;* D), where $D^{\pm}$ is the shadow obtained from D by erasing

everything except D:*prin*. D^o is called an *initial shadow* for *S*. We denote by $A_o[M(S^o, D^o)] \rightarrow s_r(M(S, D))$ such run, where s_r is the bitstring returned by A_o at the end of the run.

Accordingly, we *define* a low level run starting from *S* with (valid) shadow D against A, written $A[M(S, D)] \rightarrow s_r(M)$, as $(A_o ; A)[M(S^o, D^o)] \rightarrow s_r(M)$ where $(A_o; A)$ represents an adversary that first runs A_o and then runs A.

Definition 10.27 (Valid Shadow). Let *S* be a safe system with shadow D. We say that D is a *valid shadow* for *S* if there exist an initial safe system S^o with initial shadow $D^{\pm}$, normal transitions $S^o \xrightarrow{\varphi^o} S$, and a PPT algorithm $A_{\pm}$ such that $A_o[M(S^o, D^o)] \rightarrow public(D)(M(S, D))$, and $ms^D(\eta) \geq max_{\lceil M \rceil \leq c} \lceil [\![M^D]\!] \rceil$ where *c* is the constant given by the safety condition and $[\![M^D]\!]$ is the result of marshaling the low-level representations of *M*.

We say that D is a valid shadow for two safe systems $S_1 \cong S_2$, if the same A_o initialises both $M(S_1, D)$ and $M(S_2, D)$, and $ms^D(\eta) \geq max_{\lceil M \rceil \{c_1, c_2\}} \lceil [\![M^D]\!] \rceil$, where c_1 and c_2 are the constants given by the safety condition of S_1 and S_2 respectively.

Our first theorem expresses the completeness of our high-level transitions: every low-level attack can be described in terms of high-level transitions. More precisely, the probability that an interaction with a PPT adversary yields a machine state unexplained by any high-level transitions is negligible.

Theorem 10.4 (Completeness for Reachable States). *Let S be a safe stable system,* D *a valid shadow for S, and* A *a PPT algorithm.*

The probability that $A[M(S, D)]$ *completes and leaves the system in state* M' *with* $M' \neq M(S', D')$ *for any normal transitions* $S \xrightarrow{\varphi} S'$ *with valid shadow* D *is negligible.*

Proof Sketch. We just sketch the proof and refer the reader to Appendix C for the full construc-tions and proofs of the associated lemmas. The proof is done by tracing the cases when the behaviour of machine M(*S*, D) is not in accordance with the high-level semantics and check-ing that the probability of occurrence of such cases is negligible. A more detailed sketch is the following:

- We start by defining variants of M(*S,* D) called the defensive variants $\overline{\mathrm{M}}$ (*S,* D) (Defini-tion C.1). These machines behave like M(*S,* D) but include an extra wire where a failure signal is sent whenever the low-level interaction is not in accordance with the high-level se-mantics. The reader should be aware that these machines are just used as a proof technique, hence there is no need to implement it. All our results are stated in terms of M(*S,* D).
- The second step is to create a machine $\overline{N}^{\tilde{0}}(S,\mathrm{D})$ that behaves like M(*S,* D) but has a com-mon state for all machines $\mathrm{M}_a(S,\mathrm{D})$ (Definition C.5). This is the same as having one single machine that includes all the $\mathrm{M}_a(S,\mathrm{D})$ machines, for all $a \in H$. We show that M(*S,* D) is equivalent to $\overline{N}^{\tilde{0}}$ (*S;* D).
- The third step is to define $\overline{N}$ (*S,* D). This is the extreme version of $\overline{N}^{\tilde{n}}$ (*S,* D) where all the encrypted messages are 0's and no signing is ever performed.

Then we have two different arguments. The first is the partial completeness of $\overline{M}$ (*S,* D), $\overline{N}$ (*S,* D), and the failure of $\overline{N}$(*S,* D).

- We show that all runs of $\overline{M}$(*S,* D) and $\overline{N}$(*S,* D), where the failure signal is not sent are in confor mance with the high-level semantics (Lemma C.6 and Lemma C.7).
- We show that the probability that the failure signal is issued by N(*S;* D) machine is negligible by reducing it to the security of the encryption, authentication and signing schemes (Lemma C.8).

The second argument is that $\overline{M}$(*S,* D) is indistinguishable from $\overline{N}$(*S,* D), hence the failure of the former implies the failure of the latter, which only happens with negligible probability. This is done as follows:

- $\overline{N}^{\tilde{n}}(S,D)$ machines are parameterised by $\tilde{n} = (n_a)_{a\in H}$. This parameter defines how many messages to each honest principal will be "fake" (a fake message is one where we encrypt 0's instead of the real bitstring). Whenever n_a is reached, it starts behaving like $\overline{M}_a(S,\mathrm{D})$. For the fake messages we keep an internal table that associates the fake bitstring to the real message so that we can proceed with the correct value when the fake message is provided back to the machine. With a standard

cryptographic argument we show that distinguishing $\overline{N}^{\tilde{n}}(S, \mathrm{D})$ from $\overline{N}^{\tilde{n}+1}(S, \mathrm{D})$, where $\tilde{n}$ + 1 has all the components equal to $\tilde{n}$ except for n_a that we replace by n_a + 1 for some principal a (Lemma C.12).

- We show via a cryptographic argument that for all PPT adversaries, $\overline{M}(S, \mathrm{D})$ is indistin-guishable from $\overline{N}(S, \mathrm{D})$ (Lemma C.13, C.14, and C.15).

This concludes our proof.

Finally, main result states the soundness of equivalence: to show that the machines that implement two stable systems are indistinguishable, it suffices to show that they are safe and bisimilar. We just need an extra condition that the padding size is the same in both cases.

Theorem 10.5 (Soundness for Equivalences). *Let* S_1 and S_2 *be safe stable systems,* D *a valid shadow for both* S_1 *and* S_2*.*

If $S_1 \cong S_2$*, then* $\mathrm{M}(S_1, \mathrm{D}) \approx \mathrm{M}(S_2, \mathrm{D})$*.*

Proof Sketch. For this theorem we also refer the reader to Appendix C for the full proofs of the associated lemmas. The proof is done reusing some of the previous lemmas, in particular Lemma C.15 and with the special Lemmas C.16 and C.17. This lemmas state that for equivalent systems S_1 and S_2 the probabilities of failure of $\overline{N}(S_1, \mathrm{D})$ and $\overline{N}(S_2, \mathrm{D})$ are the same up to negligible probability.

Advancement questions

1. Why does the algebras are widely used in the study of security of concurrent systems?
2. On what stages we are able to define the high-level semantics?
3. What the CFGBLOCK and CFGFWD rules do?
4. What is the scheduler algorithm?
5. What have we do if we quantify over all local processes?
6. What protocol was used as benchmark to verivy the presented framework?
7. Name the roles of the electronic protocol.
8. What the representations for terms do we use?

9. Does the internal representation of terms uses the same grammar as in the high-level language except for atomic subterms?
10. What pieces of the system $send_b$ and $receive_b$ protocols are used?

REFERENCES

[1] Pedro Miguel dos Santos Alves Madeira Adão Formal Methods for the Analysis of Security Protocols / Pedro Miguel dos Santos Alves Madeira Adão // PhD diss., INSTITUTO SUPERIOR TÈCNICO. – 2006.

[2] Milner R. Communication and Concurrency / R. Milner. - Prentice Hall, 1989. – 256p.

[3] Lowe G. An attack on the needham-schroeder public-key authentication protocol.Information / G. Lowe. // Processing Letters. - 1995. -Vol56(3). -p131–133.

[4] Lowe G. Breaking and fixing the Needham-Shroeder public-key protocol using FDR. In Tiziana Margaria and Bernhard Steffen, editors, Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 1055 of Lecture Notes in Computer Science / G. Lowe. // Springer-Verlag. -1996. -p147–166.

[5] Abadi M. A calculus for cryptographic protocols: The Spi Calculus. Information and Computation / M.Abadi, A.D. Gordon // Full version available as SRC Research Report 149, January 1998. –January 1999, -Vol148(1). -p1–70.

[6] Milner R. Communicating and Mobile Systems : The pi-Calculus. / R.Milner. - Cambridge University Press, June 1999, -31p.

[7] Abadi M. Mobile values, new names, and secure communication / M.Abadi, Cédric Fournet // In 28th ACM Symposium on Principles of Programming Languages (POPL). -2001. -p104–115.

[8] Mart´ ın Abadi. Secure implementation of channel abstractions / Mart´ ın Abadi, Cédric Fournet, Georges Gonthier // Information and Computation. -2002. -Vol174(1). -p37–83.

[9] Blanchet B. Automated verification of selected equivalences for security protocols / B. Blanchet, M. Abadi, C. Fournet // In Proceedings of the 20th IEEE Symposium on Logic in Computer Science (LICS). – 2005. – p. 331–340.

[10] Rackoff C. and Simon D.R. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack / C. Rackoff, D.R. Simon // In Feigenbaum. – 1991. – p. 433–444.

[11] Goldwasser S. A digital signature scheme secure against adaptive chosen-message attacks / S. Goldwasser, S. Micali, and R.L. Rivest // SIAM Journal on Computing. -1988. -Vol17(2). -p281–308.

[12] Goldwasser S. Probabilistic encryption / S. Goldwasser and S. Micali // Journal of Computer and Systems Sciences. - 1984. -Vol28(2). – p. 270–299.

[13] Martin Abadi Authentication primitives and their compilation / Martin Abadi, Cédric Fournet, and Georges Gonthier // In Proceedings of the 27th ACM Symposium on Principles of Programming Languages (POPL 2000). -2000. -p302–315. ACM.

[14] Backes M. A cryptographically sound Dolev-Yao style security proof of an electronic payment system / M. Backes and M. Dürmuth // In CSFW05 [CSF05] –p. 78–93.

[15] Bellare M. iKP — a family of secure electronic payment protocols / M. Bellare, J. Garay, R. Hauser, A. Herzberg abd H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner // In Proceedings of the 1st USENIX Workshop on Electronic Commerce. - 1995.

[16] Bellarer M.. Design, implementation, and deployment of the ikp secure electronic payment system / M. Bellare, J. Garay, R. Hauser, A. Herzberg abd H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreveghen, and M. Waidner // IEEE Journal on Selected Areas in Communications. -2000. Vol18(4). –p. 611–627.

[17] Backes M. A composable cryptographic library with nested operations. In S. Jajodia, V. Atluri, and T. Jaeger, editors / M. Backes, B. Pfitzmann, and M. Waidner // Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS). -2003. –Vol15. –p. 220–230.

[18] Lincoln P. A probabilistic polynomial-time framework for protocol analysis. In M. Reiter, editor / P. Lincoln, J. C. Mitchell, M.

Mitchell, and A. Scedrov // Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS). -1998. –p. 112–121.

[19] Martin Abadir. Authentication primitives and their compilation / Martin Abadi, Cédric Fournet, and Georges Gonthier // In Proceedings of the 27th ACM Symposium on Principles of Programming Languages (POPL 2000). -2000. – p. 302–315.

CHAPTER 11. A PROCESS ALGEBRA FOR REASONING ABOUT QUANTUM SECURITY

Content of the chapter 11

Introduction

Security protocols are, in general, composed by several agents running in parallel, where each agent computes information (bounded by polynomial-time on the security parameter) and exchange it with other agents. In the context of quantum processes, the computation is bounded by quantum polynomial-time and the information exchanged is supported by qubits. In this Chapter, the problem of defining quantum security properties is addressed using a quantum polynomial-time process algebra. This approach is highly inspired in [1, 2, 3].

The process algebra is introduced together with the logarithmic cost random access machine. Both the syntax and the semantics of the process algebra are clearly established, and the section is concluded by presenting the notion of observational equivalence. Sections are devoted to emulation and its composition theorem, and quantum zero-knowledge is defined using process emulation.

11.1 Process Algebra

In the context of security protocols it is common to consider a security parameter $\eta \in \mathrm{N}$. In the case of quantum protocols we will also consider such parameter in order to bound the quantum complexity of the principals and adversaries. From now on, the symbol η is reserved to designate such security parameter. The role of this parameter is twofold: it bounds to a polynomial on η the number of qubits that can be sent through channels, and it bounds all the computation to quantum polynomial time (on η). We now detail these aspects culminating with the presentation of the process algebra language.

11.2 Quantum polynomial machines

The computational model we adopted to define quantum polynomial machine is based on the logarithmic cost random access machine [4] and it is quite similar to the quantum random access machine in [5]. We consider a hybrid model using both classic and quantum memory.

In order to cope with a countable set of qubits *qB* we adopt the following Hilbert space *H* (isomorphic to $\ell^2\left(2^{qB}\right)$ and $L^2(2^{qB}, \#)$) to model the quantum state (see [6, 7] for a discussion on why *H* is the correct Hilbert space for modelling a countable set of qubits):

- each element is a map $|\psi\rangle : 2^{qB} \to C$ such that:
- supp $\left(|\psi\rangle\right) = \left\{\upsilon \in 2^{qB} : |\psi\rangle(\upsilon) \neq 0\right\}$ is countable;
- $\sum_{\upsilon \in 2^{qB}} ||\psi\rangle(\upsilon)|^2 = \sum_{\upsilon \in \mathrm{supp}(|\psi\rangle)} \left||\psi\rangle(\upsilon)\right|^2 < \infty$;
- $|\psi_1\rangle + |\psi_2\rangle = \lambda\upsilon.|\psi_1\rangle(\upsilon) + |\psi_2\rangle(\upsilon)$;
- $z|\psi\rangle = \lambda\upsilon._z|\psi\rangle(\upsilon)$;
- $\langle\psi_1|\psi_2\rangle = \sum_{\upsilon \in V} \overline{|\psi_1\rangle(\upsilon)}|\psi_2\rangle(\upsilon)$.

The inner product induces the norm $\left\||\psi\rangle\right\| = \sqrt{\langle\psi|\psi\rangle}$ and so, the distance $d\left(|\psi_1\rangle, |\psi_2\rangle\right) = \left\||\psi_1\rangle - |\psi_2\rangle\right\|$. Clearly, $\left\{|\upsilon\rangle : \upsilon \in 2^{qB}\right\}$ is an orthonormal basis of *H* where $|\upsilon\rangle(\upsilon) = 1$ and $|\upsilon\rangle(\upsilon') = 0$ for every $\upsilon' \neq \upsilon$. This basis is called the computational or logic basis of *H*.

A configuration of a quantum random access machine (QRAM) is triple ξ = (m, $|\psi\rangle$, *s)* where $m \in \mathrm{N}^{\mathrm{N}}$. $|\psi\rangle \in$ *H* and $s \in \mathrm{N}$. The first component of the triple represents the classical memory of the machinean infinite sequence of natural numbers, the second component represents the quantum state of the machine, and finally the third component is a counter that indicates how many (qu)bit operations are allowed.

We associate to each QRAM a positive polynomial *q* for bounding the number of allowed (qu)bit operations to q(η). In this way, we force each QRAM to terminate in polynomial-time. Given a finite set of qubits at state $|\psi\rangle$, the *intiial configuration* of the QRAM is the triple

$\xi_0(|\psi\rangle)$ = (m_0, $|\psi\rangle \otimes |0\rangle$, q(η)), where the sequence m_0 is such that $m_0(k) = 0$ for all $k \in \mathrm{N}$ and $|0\rangle$ is the unit vector in H such that $|0\rangle(\emptyset) = 1$ (note that if Q is a 2^n dimension Hilbert space, then there is a canonical isomorphism between H and $Q \in H$, and therefore $|\psi\rangle \otimes |\vec{0}\rangle \in Q \otimes H$ can be seen as a unit vector in H). A QRAM receives as input a finite sequence of qubits, but since it is always possible to encode classical bits in qubits this is not a limitation.

The set of atomic commands $\widetilde{A}\widetilde{C}$, and their associated cost is presented in the table below.

!!!

Number	Instruction	Computational cost
1	R_i=n	n|
2	R_i= R_j	|R_j|
3	R_i= R_j+ R_k	|R_j|+|R_k|
4	R_i= R_j - R_k	|R_j|+|R_k|
5	R_i= R_j R_k	|R_j|×|R_k|
6	R_i= R_j / R_k	|R_j|×|R_k|
7	R_i= R_{Rj}	|R_j|+|R_{Rj}|
8	R_{Rj} = R_j	|R_j|+|R_j|
9	Pauli$_x$[b]	1
10	Pauli$_y$[b]	1
11	Pauli$_z$[b]	1
12	Hadamard[b]	1
13	phase[b]	1
14	$\frac{\pi}{8}$[b]	1
15	c-not[b_1, b_2]	1
16	measure[b]→R_i	1

Most of the commands above are self-explanatory, but it is worthwhile to notice that all commands are deterministic with exception of measure. Indeed, according to the measurement postulates of quantum mechanics (see for instance [8]), when a quantum system is

measured the outcome is stochastic, and moreover the state evolves accordingly to this outcome. Note that we only consider measurements over the computational basis, nevertheless this is not a limitation since any other qubit measurement can be performed by applying a unitary transformation before measuring the qubit over the computational basis.

The set of QRAM *commands* $\tilde{C}$ is obtained inductively as follows:

- $\alpha \in \tilde{C}$ if $\alpha \in \tilde{A}\tilde{C}$;
- $c_1; c_2 \in \tilde{C}$ if $c_1, c_2 \in \tilde{C}$;
- (if ($R_n > 0$) then c) $\in \tilde{C}$ if $c \in \tilde{C}$;
- (while ($Rn > 0$) c) $\in \tilde{C}$ if $c \in \tilde{C}$.

The *execution* of a QRAM command c is a stochastic function between configurations. Let $\Xi = \mathrm{N}^{\mathrm{N}} \times H \times \mathrm{N}$ be the set of all configurations, and $\mathrm{Prob}_{\mathrm{fin}}(\Xi)$ be the set of all probability measures over $(\Xi, 2^{\Xi})$ such that only a finite set of configurations have probability different from 0. The execution of a QRAM command c is a map $\mathrm{run}_c : \Xi \to \mathrm{Prob}_{\mathrm{fin}}(\Xi)$, and we write $[c]\xi \to_p \xi'$ to denote that $\Pr_{\mathrm{run}(\xi)}(\xi') = p$. The execution of QRAM commands can be defined using the following rules, which are quite intuitive:

$$\frac{s \geq |n|}{[R_i = n](m, |\psi\rangle, s) \to_1 (m', |\psi\rangle, s - |n|)}(R_i = n),$$

where $\mathrm{m}'(k) = m(k)$ *for all* $k \neq i$ *and* $\mathrm{m}'(i) = n$;

$$\frac{s \geq |R_j|}{[R_i = R_j](m, |\psi\rangle, s) \to_1 (m', |\psi\rangle, s - |R_j|)}(R_i = R_j),$$

where $\mathrm{m}'(k) = m(k)$ *for all* $k \neq i$ *and* $\mathrm{m}'(i) = \mathrm{m}(j)$;

$$\frac{s \geq |R_j| + |R_k|}{[R_i = R_j + R_k](m, |\psi\rangle, s) \to_1 (m', |\psi\rangle, s - (|R_j| + |R_k|))}(R_i = R_j + R_k)$$

where $\mathrm{m}'(k) = m(k)$ *for all* $k \neq i$ *and* $\mathrm{m}'(i) = \mathrm{m}(j) + \mathrm{m}(k)$;

$$\frac{s \geq |R_j| + |R_k|}{[R_i = R_j - R_k](m, |\psi\rangle, s) \to_1 (m', |\psi\rangle, s - (|R_j| + |R_k|))}(R_i = R_j - R_k)$$

where $\mathrm{m}'(k) = m(k)$ *for all* $k \neq i$ *and* $\mathrm{m}'(i) = max(\mathrm{m}(j) - \mathrm{m}(k), 0)$;

$$\frac{s \geq |R_j| \times |R_k|}{[R_i = R_j R_k](m, |\psi\rangle, s) \rightarrow_1 (m', |\psi\rangle, s - (|R_j| \times |R_k|))} (R_i = R_j R_k),$$

where m′*(k) = m(k) for all k ≠ i and* m′*(i)* = m*(j)*m*(k);*

$$\frac{s \geq |R_j| \times |R_k|}{[R_i = R_j / R_k](m, |\psi\rangle, s) \rightarrow_1 (m', |\psi\rangle, s - (|R_j| \times |R_k|))} (R_i = R_j / R_k),$$

where m′*(k) = m(k) for all k ≠ i and* m′*(i) = [*m*(j)/*m*(k)];*

$$\frac{s \geq |R_j| + |R_{R_j}|}{[R_i = R_{R_j}](m, |\psi\rangle, s) \rightarrow_1 (m', |\psi\rangle, s - (|R_j| + |R_{R_j}|))} (R_i = R_{R_j}),$$

where m′*(k) = m(k) for all k ≠ i and* m′*(i) = m(*m*(j));*

$$\frac{s \geq |R_i| + |R_j|}{[R_{R_j} = R_j](m, |\psi\rangle, s) \rightarrow_1 (m', |\psi\rangle, s - (|R_i| + |R_j|))} (R_{R_j} = R_j),$$

where m′*(k) = m(k) for all k ≠ m(i) and* m′*(m(i))* = m*(j);*

$$\frac{s \geq 1}{Pauli_X[b](m, |\psi\rangle, s) \rightarrow_1 (m, |\psi'\rangle, s - 1)} (Pauli_X[b]),$$

where $|\psi'\rangle$ is obtained from $|\psi\rangle$ by appluing the Pauli$_X$ operator $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ on qubit b. Similar rules apply to the following one-qubit operators:

$$Pauli_Y \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \; Pauli_Z \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix};$$

$$Hadamard \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \; Phase \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \; \frac{\pi}{8} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix};$$

$$\frac{s \geq 1}{[c - not[b_1, b_2]](m, |\psi\rangle, s) \rightarrow_1 (m, |\psi'\rangle, s - 1)} (c - not[b_1, b_2]),$$

where $|\psi'\rangle$ is obtained from $|\psi\rangle$ by applying the control-not operator

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

on qubits b_1 and b_2;

$$\frac{s \geq 1}{[measure[b] \to R_i](m, |\psi\rangle, s) \to_p (m', |\psi'\rangle, s-1)} (measure[b] \to R_i = 0)$$

where $|\psi'\rangle$ is equal to $\frac{P_0|\psi\rangle}{|P_0|\psi\rangle|}$, $p = |P_0|\psi\rangle|$ is (P_0 the projector onto the subspace of H where qubit b takes value $|0\rangle$), m'(i) = 0 and m'(j) = m(j) for all $j \neq$ i;

$$\frac{s \geq 1}{[measure[b] \to R_i](m, |\psi\rangle, s) \to_p (m', |\psi'\rangle, s-1)} (measure[b] \to R_i = 1)$$

where $|\psi'\rangle$ is equal to $\frac{P_1|\psi\rangle}{|P_1|\psi\rangle|}$, $p = |P_1|\psi\rangle|$ is (P_1 the projector onto the subspace of H where qubit b takes value $|1\rangle$), m'(i) = 1 and m'(j) = m(j) for all $j \neq$ i;

$$\frac{[c_1](m, |\psi\rangle, s) \to_{p1} (m', |\psi'\rangle, s') \to_{p2} (m'', |\psi''\rangle, s'')}{[c_1; c_2](m, |\psi\rangle, s) \to_{p1 \times p2} (m'', |\psi''\rangle, s'')} ([c_1; c_2]);$$

$$\frac{m(n) > 0 \;\; s \geq |R_n| \;\; [c] \; (m, |\psi\rangle, s - |R_n|) \to_p (m', |\psi'\rangle, s')}{[if \; (R_n > 0) \; then \; c](m, |\psi\rangle, s) \to_p [c](m', |\psi'\rangle, s')} (ifT);$$

$$\frac{m(n) = 0}{[if \; (R_n > 0) \; then \; c](m, |\psi\rangle, s) \to_1 (m, |\psi\rangle, s)} (if \; \perp);$$

$$\frac{[c;\ (while\ (R_n > 0)c)](m,|\psi\rangle, s-|R_n|) \rightarrow_p (m',|\psi'\rangle, s')}{[(while\ (R_n > 0)c)](m,|\psi\rangle, s) \rightarrow_p (m',|\psi'\rangle, s')}(whileT);$$

$$\frac{m(n) = 0}{[(while\ (R_n > 0)c)](m,|\psi\rangle, s) \rightarrow_p (m,|\psi\rangle, s-|R_n|)}(while\ \perp).$$

Observe, that the reduction of QRAM commands always terminate, since every computation is bounded by $q(\eta)$ (qu)bit steps. The execution of a QRAM command can be seen as a word run of a quantum automata [9], however a detailed discussion about this subject is out of the scope of this abstract.

The output of a QRAM is the quantum state of a set of qubits. This output set is determined by another positive polynomial o associated to the machine. Given a security parameter n, the set of output qubits is constituted by the first $o(\eta)$ qubits.

Definition 11.1. A *quanlum polynomil machine* is a triple $M = (c, q, o)$ where c is a QRAM command, q is a positive step bounding polynomial and o is a positive output polynomial. We denote the set of all these triples by QPM.

Given a quantum polynomial machine M and a security parameter η, the computation of M over state $|\psi\rangle$ is the probability distribution over the state of the first $o(\eta)$ qubits of $|\psi'\rangle$, where this distribution is defined by the execution rules $[c](m_0,|\psi\rangle, q(\eta)) \rightarrow_p (m',|\psi'\rangle, s')$. Hence, the computation of a QRAM is a probability distribution over the state space of the first $o(\eta)$ qubits. It is traditional in quantum algorithms to measure all relevant qubits at the end of the computation in order to obtain a classical result (see Shor's and Grover's algorithms). However, since we use QRAM to compute quantum information that can be sent through quantum channels, we do not impose this final measurement since it may be desirable to send a superposition through a quantum channel.

The following result asserts that the QRAM model is equivalent to the usual quantum circuit computational model (a careful presentation of this result is out of the scope of this abstract).

Proposition 11.1. For any uniform family of polynomial quanlum circuils $Q = \{Q_\eta\}_{\eta \in N}$, there exists a quanlum polynomial machine M_Q such that lhe M_Q compules the same slochastic function as Q. Moreover, for any quanlum polynomial machine M there exists an equivalenl uniform family of polynomial quanlum circuils $Q_M = \{Q_n\}_{\eta \in N}$.

Proof. Proof (Sketch): Note that a uniform circuit uses precisely the gates defined as quantum atomic commands of the QRAM. The construction of the circuit can be mimicked by a RAM command c. Since this construction must be polynomial in η, the program must terminate in polynomial time and therefore, there is a polynomial q to bound the number of steps, finally the output must always be a polynomial set of qubits, and therefore we are able to construct an equivalent QRAM machine.

On the other hand a QRAM program is the realisation of the uniform family construction, since, for each η, a circuit can be retrieved by looking at the finite (do not forget that QRAM programs always terminate) sequence of quantum atomic gates generated by the execution of the command. The stochastic nature of the execution does not bring a problem, since gates placed after a measurement can be controlled by the outcome of that measurement. If a measurement gives the value 1 to a qubit and in that case a gate U is placed at some qubit b, then the circuit should be constructed by placing a control-U gate controlled by the measured qubit and targeted at b.

11.3 Process algebra

As stated before, we require to know who possesses a qubit in order to know who can retrieve some piece of information. In order to deal with this fact, a qubit is considered to belong to some agent, and therefore, the set of qubits qB is partitioned among all agents. To make this more precise, a countable set $A = \{a_1, \ldots, a_k, \ldots\}$ of agents is fixed once and for all, and moreover the partition $\tilde{qB} = \{qB\text{ai}\}_{a_i \in A}$ of qB is such that each set qBai is countable and recursively enumerable.

Note that each qBai has a total order (with a bottom element) induced by its recursive enumeration. The purpose of this total ordering

is to reindex the qubits accessed by a QPM *M* when an agent *a* executes *M*. An obvious desideratum of the system is that an agent *a* is restricted to compute over its own qubits qB_a, and therefore, when agent *a* executes a quantum polynomial machine *M*, this machine must have access only to the qubits in qB_a (note that if the qubits of *a* are entangled with other qubits, then when the former are modified so can be the latter). Therefore, if, for instance, an agent *a* executes a machine that consists of the command $\text{Pauli}_X[b]$, and if qB_a is recursively enumerated by γ, then the command effectively executed is $\text{Pauli}_X[\gamma(b)]$. The same procedure applies to the input and output qubits, so when a machine executed by *a* outputs the first $o(\eta)$ qubits, the machine is in fact outputting the qubits $\{\gamma(o(1)),...,\gamma(o(\eta))\} \subset qB_a \subseteq qB$.

Communication between agents is achieved via public channels, allowing qubits to be exchanged. Clearly, this process is modelled by modifying the partition of *qB*. It is also convenient to allow parallelism inside an agent (that is, an agent may be constituted by several processes in parallel), for this purpose, private channels (that cannot be intercepted) allowing communication between the agent local processes are introduced. To make this assumptions clear, two countable disjoint sets of *quantum channels* are considered, the set of *global or public channels* $G=\{g_1, g_2, \ldots, g_k, \ldots\}$, and the set of *local or priate channels* $L=\{l_1, l_2, \ldots, l_k, \ldots\}$. We denote by *C* the set *G* U L. All global channels can be read and written by an adversary while local channels correspond to private communication from one agent to itself. One role of the security parameter is to bound the bandwidth of the channels. Hence, we introduce a *bandwidth map* $\text{bw} : C \rightarrow \text{q}$, where q is the set of all polynomials taking positive values. Given a value η for the security parameter, a channel *c* can send at most $\text{bw}(c)(\eta)$ qubits.

We also consider a countable set of variables $Var=\{x_L, x_2, \ldots, x_k, \ldots\}$, which are required to define qubit terms. A qubit term *t* is either a finite subset of *qB* or a variable $x \in Var$.

Finally, we present the language of processes, which is a fragment of π-calculus. Mind that the overall computation must be quantum polynomial on η and therefore we do not cope with recursion nor

mobility. First, we establish the language of an agent, that we call local process language.

Definition 11.2. The language of local processes L is obtained inductively as follows:

- $0 \in$ L (termination);
- $c\ \langle M(t)\rangle \in L$ where M $\in$ QPM, t is a qubit term, and $c \in C$ (output);
- $c(x).Q \in L$ where $c \in C, x \in Var$ and $Q \in L$ (input);
- $[M(t)=0]Q$ where $M \in QPM$, t is a qubit term, and $Q \in L$ (match);
- (Q_1, Q_2) where $Q_1, Q_2 \in L$ (parallel composition);
- !qQ where $Q \in L$ and $q \in$ q (bounded replication).

Most of the (local) process terms are intuitive. The output term $c\langle M(qB')\rangle$ means that the output of machine *M*, which received the finite set of qubits *qB'* as input, is sent through channel c. The input term *c(x).Q* means that a set of qubits is going to be received on c, and upon reception, *x* takes the value of the received qubits.

After fixing the security parameter η, we can get rid of replication by evaluating each process $!_q R$ as $q(\eta)$ copies of *R* in parallel. Therefore, we always assume that a process term has no replication. Now, as state before, a protocol is constituted by a set o agents running in parallel, therefore the global language (or protocol language) is quite simple:

Definition 11.3. The *language of global processes* $\acute{G}$ over a set of agents *A* is defined inductively as follows:

- $0 \in \acute{G}$ (global termination) ;
- $P \| (a : Q) \in \acute{G}$ where $P \in \acute{G}$, $a \in A$ does not occur in *P*, and $Q \in L$ (global parallel composition).

The following example uses the process language to describe the RSA cryptanalysis using Shor's algorithm.

Example 11.1 (Shor's based RSA cryptanalysis). Let *p, q* be primes (with η length binary expansion), and *e, d* integers such that $ed = 1 \mod \phi(pq)$. Alice is a simple process *A* that knows some message *w*

and outputs w^e mod pq, where e is the public key of Bob. This dummy process can be presented as

$$(a : A(\omega)) := \left(a : g\left\langle \omega^e \bmod pq \right\rangle\right).$$

Bob receives x and computes x^d mod pq. This procedure can be modelled by the following process:

$$(b : B) := \left(b : g(x).\left(l\left\langle x^d \bmod pq \right\rangle \mid l(y).0\right)\right).$$

Therefore the RSA protocol is given by the process $(a : A(\omega)) \| (b : B)$. Finally, we can write the "attacking" process, Eve. She factorises pq, inverts e mod $\phi(pq)$ (thus, allowing her to find d), and intercepts the message sent by Alice (on channel g). We write this process as follows:

$$\left(c : l_1\left\langle Shor(pq)\right\rangle \mid l_1(y).l_2\left\langle Inv(y,e)\right\rangle \mid g(x).l_2(z).\left(l_3\left\langle x^z \bmod pq\right\rangle \mid l_3(\omega).0\right)\right)$$

11.4 Semantics

In order to define the semantics of a local process we need to introduce the notion of local configuration. A *local configuration* or *agent configuration* is a triple ($|\psi\rangle$, qB_a, Q) where $|\psi\rangle \in H$, $qB_a \subseteq qB$ is a countable, recursive enumerable set and $Q \in L$. The first element of the local configuration is the global state of the protocol, the second element is the set of qubits the agent possesses and the last element is the local process term.

The semantics of a local process is a probabilistic transition system where the transitions are defined by rules. We use $(|\psi\rangle, \mathrm{qB_a}, \mathrm{Q}) \rightarrow_{\mathrm{p}} (|\psi'\rangle, \mathrm{qB_a}, \mathrm{Q}')$ to state that, at global state $|\psi\rangle$), when agent a possesses qubits qB_a, the local process Q is reduced to Q' and global state is modified to $|\psi'\rangle$ with probability p. It is also worthwhile to observe that we use the notation $M\left(|\psi\rangle, qB_a, qB_1\right) \rightarrow_p \left(|\psi'\rangle, qB_2\right)$ to denote that the execution of the QRM M, operating on qB_a (that is, using the recursive enumeration of qB_a to reindex the position of the qubits), and receiving as input qB_1,

outputs qB_2 and modifies the global state $|\psi\rangle$ to $|\psi'\rangle$ with probability p. For the case of local processes, the sets qB_1 and qB_2 are irrelevant, because the qubits owned by the agent remain the same when a local communication (LCom rule) is applied. Their functionality will be clear when we present the global rules.

$$\frac{M(|\psi\rangle, qB_a, qB_1) \to_p (|\psi'\rangle, qB_2)\ qB_1, qB_2 \in qB_a\ |qB_a| \le bw(l)(\eta)}{(|\psi\rangle, qB_a, l(x).Q \mid l\langle M(qB_1)\rangle) \to_p (|\psi'\rangle, qB_a, Q^x_{qB_a})} (LCom)$$

We also introduce the term M; Meas to denote the machine that, after executing M performs a measurement on the computational basis of the output qubits of M. So a match corresponds to performing a measurement on the output qubits of M and checking whether the result is the 0 word.

$$\frac{(M; Meas)(|\psi\rangle, qB_a, qB_1) \to_p (|\psi'\rangle, qB_2)\ |\psi'\rangle_{qB_2} = |\vec{0}\rangle}{(|\psi\rangle, qB_a, [M(qB_1) = 0].Q) \to_p (|\psi'\rangle, qB_a, Q)} (MatchT)$$

$$\frac{(M; Meas)(|\psi\rangle, qB_a, qB_1) \to_p (|\psi'\rangle, qB_2)\ |\psi'\rangle_{qB_2} \neq |\vec{0}\rangle}{(|\psi\rangle, qB_a, [M(qB_1) = 0].Q) \to_p (|\psi'\rangle, qB_a, 0)} (Match\ \bot)$$

The remaining rules are self-explanatory.

$$\frac{(|\psi\rangle, qB_a, P) \to_p (|\psi'\rangle, qB_a, P')}{(|\psi\rangle, qB_a, P \mid Q) \to_p (|\psi'\rangle, qB_a, P' \mid Q)} (LLPar)$$

$$\frac{(|\psi\rangle, qB_a, Q) \to_p (|\psi'\rangle, qB_a, Q')}{(|\psi\rangle, qB_a, P \mid Q) \to_p (|\psi'\rangle, qB_a, P \mid Q')} (LRPar)$$

We proceed by presenting the global rules. A *global configuration* is a triple $(|\psi\rangle, q\tilde{B}, P)$ where $|\psi\rangle \in H$, $q\tilde{B} = \{qB_a\}_{a \in A}$ is a partition of qB indexed by the set of agents A (where each qB_a is countable and r.e.) and $P \in \acute{G}$. The semantics of a global process is defined by the following rules:

$$\frac{(|\psi\rangle, qB_a, Q) \to_p (|\psi'\rangle, qB_a, Q')}{(|\psi\rangle, q\tilde{B}, (a:Q)) \to_p (|\psi'\rangle, q\tilde{B}, (a:Q))} (LtoG)$$

$$\frac{M\left(\left|\psi\right\rangle, qB_a, qB_1\right) \rightarrow_p \left(\left|\psi'\right\rangle, qB_2\right)\ qB_1, qB_2 \in qB_b\ \left|qB_2\right| \le bw(g)(\eta)}{\left(\left|\psi\right\rangle, q\tilde{B}, (a : g(x).Q) \,\|\, (b : g\langle M(qB_1)\rangle)\right) \rightarrow_p \left(\left|\psi'\right\rangle, q\tilde{B}', (a :\ Q^{x}_{qB_2})\right)} (GCom)$$

where $q\tilde{B}' = \{qB'_a\}_{a\in A}$, $qB'_a = qB_a \bigcup qB_2$, $qB'_b = qB_b \setminus qB_2$, and $qB'_c = qB_c$ for all $c \neq a, b$.

$$\frac{\left(\left|\psi\right\rangle, q\tilde{B}, P_1\right) \rightarrow_p \left(\left|\psi'\right\rangle, q\tilde{B}', P'_1\right)}{\left(\left|\psi\right\rangle, q\tilde{B}, P_1 \,\|\, P_2\right) \rightarrow_p \left(\left|\psi'\right\rangle, q\tilde{B}', P'_1 \,\|\, P_2\right)} (GLPar)$$

$$\frac{\left(\left|\psi\right\rangle, q\tilde{B}, P2\right) \rightarrow_p \left(\left|\psi'\right\rangle, q\tilde{B}', P'_2\right)}{\left(\left|\psi\right\rangle, q\tilde{B}, P_1 \,\|\, P_2\right) \rightarrow_p \left(\left|\psi'\right\rangle, q\tilde{B}', P_1 \,\|\, P'_2\right)} (GRPar)$$

All the rules are very simple to grasp. The only non trivial rule is global communication (GCom), that makes qubits to be exchanged from one agent to another, and therefore an adjustment is required in the qubit partition.

Process term reductions are non-deterministic, in the sense that several different reductions could be chosen at some step. In order to be possible to make a quantitative analysis, this reduction should be probabilistic. For the sake of simplicity, we assume a uniform scheduler, that is, the choice on any possible reduction is done with uniform probability over all possible non- deterministic reductions. We do not present in detail the scheduler model but, in principle, any probability distribution modelled by a QPM can be used to model the scheduler policy. Finally, note that by applying local and global rules, and assuming a uniform scheduler, one can define the many step reduction $\rightarrow^*_p$ such that $\left(\left|\psi_1\right\rangle, q\tilde{B}_1, P_1\right) \rightarrow^*_p \left(\left|\psi_n\right\rangle, q\tilde{B}_n, P_n\right)$, whenever:

$\left(\left|\psi_1\right\rangle, q\tilde{B}_1, P_1\right) \rightarrow_{p1} \left(\left|\psi_2\right\rangle, q\tilde{B}_2, P_2\right) \rightarrow_{p2} \cdots \rightarrow_{p_{n-1}} \left(\left|\psi_n\right\rangle, q\tilde{B}_n, P_n\right)$;

$p = \frac{p_1}{R_1} \times \frac{p_2}{R_2} \times \cdots \times \frac{p_{n-1}}{R_{n-1}}$ where *Ri* is the number of possible non-deterministic choices for $\left(\left|\psi_i\right\rangle, q\tilde{B}_i, P_i\right)$ for all $i \in \{1, \ldots, n-1\}$;

$\left(\left|\psi_n\right\rangle, q\tilde{B}_n, P_n\right)$ cannot be reducted any more.

The many step reduction takes into account the scheduler choice, by weighting each stochastic reduction p_i with yet another probability $\frac{1}{R_i}$, where R_i is the number of possible non-deterministic choices at step i.

11.4 Observations and observational equivalence

At the end of a protocol, each agent $a \in A$ is allowed to measure a polynomial (in η) number of qubits in qB_a to extract information. We can always assume that these qubits are the first, say, $r(\eta)$ qubits of qB_a where r is a positive polynomial. Therefore, the many step reduction of a process term P induces a probability distribution on $2^{r(\eta)}$, where $2^{r(\eta)}$ is the set of all possible outcomes of $r(\eta)$ qubits when measured over the computational basis (that is, $2^{r(\eta)}\ i$ is the set of all r(η)-long binary words).

Definition 11.4. Given a positive polynomial r and a global configuration $\left(|\psi\rangle, q\tilde{B}, P\right)$, let

$$\Gamma_{(|\psi\rangle, qB, P)} = \left\{\left(|\psi'\rangle, q\tilde{B}', P'\right) : \left(|\psi\rangle, q\tilde{B}, P\right) \rightarrow_p^* \left(|\psi'\rangle, q\tilde{B}', P'\right) \text{ and } p > 0\right\}$$

We define the *observation of an agent a* to be the family of probability measures

$$O_r^q = \left\{\left(2^{r(\eta)}, 2^{2^{r(\eta)}}, \Pr_{r(\eta)}^a\right)\right\}\ \eta \in \mathrm{N}$$

where:

$\Pr_{r(\eta)}^a(\{\omega\}) = \sum_{\gamma \in \Gamma_{(|\psi\rangle, qB, P)}} p_\gamma \times \left|\langle \omega | \psi_\gamma \rangle\right|$;

p_γ is such that $\left(|\psi\rangle, q\tilde{B}, P\right) \rightarrow_{p_\gamma}^* \gamma$;

$|\psi\rangle$ is the first component of γ;

$\left|\langle \omega | \psi_\gamma \rangle\right|$ is the probability of observing the r(η)-long binary word ω by measuring the $r(\eta)$ first qubits of qB_a (qubits in possession of agent *a)* of $|\psi_\gamma\rangle$*)* in the computational basis.

Note that the summation used to compute $\Pr^{a}_{r(\eta)}(\{\omega\})$ is well defined, since $\Gamma_{(|\psi\rangle,qB,P)}$ is finite. It is clear at this point,that an observation of an agent is a random r(η)-long binary word, with distribution given by $\Pr^{a}_{r(\eta)}$.

The notion of observational equivalence we adopt is based on computational indistinguishability as usual in the security community [2]. First, we introduce the concept of context. The set of *global contexts* $\tilde{C}$ is defined inductively as follows: $[\,]\in\tilde{C}$; $C[\,]\| P$ and $P\| C[\,]\in\tilde{C}$ provided that $C[\,]\in\tilde{C}$ and $P\in$Ġ. Given a context $C[\,]$ and a global process *P*, the notation $C[P]$ means that we substitute the process *P* for the $[\,]$ in $C[\,]$.

Definition 11.5. Let *P* and *P′* be process terms. We say that *P* is computationally indistinguishable by agenl a from *P′* if and only if for every context $C[\,]$, polynomials *q* and $r,|\psi\rangle\in H$, partition $q\tilde{B}$ of *qB, η* sufficiently large and binary word $\omega\in 2^{r(\eta)}$,

$$\left|\Pr^{a}_{r(\eta)}(\omega)-\Pr'^{a}_{r(\eta)}(\omega)\right|\le\frac{1}{q(\eta)}$$

where $\Pr^{a}_{r(\eta)}$ is given by the observation of *a* for configuration $\left(|\psi\rangle,q\tilde{B},C[P]\right)$ and $\Pr'^{a}_{r(\eta)}$ is given by the observation of *a* for configuration $\left(|\psi\rangle,q\tilde{B},C[P']\right)$. In such case we write $P\approx P'$.

Two processes are computationally indistinguishable if they are indistinguishable by contexts, that is, for any input (here modelled by $|\psi\rangle$ and $q\tilde{B}$*)*, there is no context which can distinguish, up to a negligible function, the outputs produced. The definition above extends the classical definition of computational indistinguishability to the quantum case, since processes can be modelled by quantum polynomial machines and therefore $C[\,]$ induces the required distinguishing

machine. A detailed proof of this result is out of the scope of this extended abstract. In order to set up compositionality, the following result is of the utmost importance:

Proposition 11.2. Computational indistinguishabilily is a congruence relation with respecl to the parallel primitive of Ǵ.

Proof. Both symmetry and reflexivity are trivial to check. Transitivity follows by triangular inequality, and taking into account that $\frac{1}{2}q(n)$ is a polynomial. Congruence on the global parallel operator follows by noticing that for any contexts $C[\ ]$ and $C'[\ ]$, $C'[C[\ ]]$ is also a context. □

11.6 Emulation and Composition Theorem

One of the most successful ways for defining secure concurrent cryptographic tasks is via process emulation [10, 11]. This definitional job boils down to the following: a process realises a cryptographic task if and only if it emulates an ideal process that is known to realise such task. In this section, guided by the goal of defining secure functionalities, we detail the notion of emulation for the quantum process calculus defined in the previous section.

Let *I* be an ideal protocol that realises (the honest part of) some secure protocol and *P* a process that implements the functionality specified by *I*. The overall goal is to show that *P* realises, without flaws, (part of) the secure functionality specified by *I*. The goal is achieved if for any real adversary, say *(a* : A), the process *P*||(a : A) is computationally indistinguishable by the adversary *a* from the process *I*||(a : *B*) for some ideal adversary *(a* : *B*), where an ideal adversary is an adversary which cannot corrupt *I* and a real adversary is any local process for agent *a*. This property asserts that given a real adversary *(a* : A), agent *a* cannot distinguish the information leaked by *P*||(a : A) from the information leaked by the well behaved process *I*||(a : B) for some ideal adversary *(a* : B), and therefore, we infer that *P*||(a : A) is also well behaved. This discussion leads to the concept of emulation with respect to a set of real adversaries $\tilde{A}$ and ideal adversaries $\tilde{B}$.

Definition 11.6. Let *P* and *I* be process terms and $\tilde{A}$ and $\tilde{B}$ sets of global processes where the only agent is the adversary *a*, then *P emulates I* with respect to $\tilde{A}$ and $\tilde{B}$ if and only if for all processes *(a* : A) $\in \tilde{A}$ there exists a process *(a* : B) $\in \tilde{B}$ such that $P\|(a : A) \approx I\|(a : B)$. In such case we write $P \equiv^{a}_{\tilde{A},\tilde{B}} I$ and say that *P* is a secure implementation of *I* with respect to $\tilde{A}$ and $\tilde{B}$.

A desirable property of the emulation relation is the so called Composition Theorem. This result was first discussed informally for the classical secure computation setting in [12], and states the following: if *P* is a secure implementation of part *I* of an ideal protocol, *R* and *J* are two protocols which use the ideal protocol *I* as a component, and finally, *R* is a secure implementation of *J*, then R^{I}_{P} should be a secure implementation of *J*. This result is captured as follows:

Theorem 11.3. Let P,I be processes, *R[ ]* and J *[ ]* contexts and $\tilde{A}$, $\tilde{B}$ sets of processes over agent a and $\tilde{C}$, $\tilde{D}$ sets of processes over agent b. If $R[I \| (a:B)] \equiv^{b}_{\tilde{C},\tilde{D}} J[I \| (a:B)]$ for any (a *: B)* $\in \tilde{B}$ and $P \equiv^{a}_{\tilde{A},\tilde{B}} I$ then for any adversary (a *: A)* $\in \tilde{A}$ there exists (a *: B)* $\in \tilde{B}$ such *that* $R[Q \| (a:A)] \equiv^{b}_{\tilde{C},\tilde{D}} J[I \| (a:B)]$.

Proof. Let *(a* : A) $\in \tilde{A}$ and *(a* : *B*) $\in \tilde{B}$ be such that $P \| (a:A) \approx I \| (a:B)$. Now choose some (b : C) $\in \tilde{C}$, clearly, $R[Q \| (a:A)] \| (c:C) \approx R[I \| (a:B)] \| (c:C)$ since $\approx$ is a congruence relation. Moreover, since $R[I \| (a:B)] \equiv_{\tilde{C},\tilde{D}} J[I \| (a:B)]$, there is a (b : D*)* $\in \tilde{D}$ such that $R[Q \| (a:B)] \| C \approx J[I \| (a:B)] \| (b:D)$. Finally, by transitivity of $\approx$**,** we have that $R[Q \| (a:A)] \| (b:C) \approx J[I \| (a:B)] \| (b:D)$ and hence $R[Q \| (a:A)] \equiv_{\tilde{C},\tilde{D}} J[I \| (a:B)]$.

Observe that ideal protocols are constituted by a honest part *I* and an ideal adversary *(a* : B), and therefore are of the form *I*‖(a : B). This justifies why $R[I \parallel (a:B)]$ was considered in the proposition above instead of $R[I]$. Moreover, adversaries for the functionality implemented by *R* and *J* might be different from those of *I* and Q, therefore, two pairs of sets of processes $\widetilde{C}$, $\widetilde{D}$ and $\widetilde{A}$, $\widetilde{B}$ are required to model two kinds of adversaries.

11.7 Quantum Zero-Knowledge Proofs

An interactive proof is a two party protocol, where one agent is called the *prover* and the other is called the *verifier*. The main objective of the protocol is to let the prover convince the verifier of the validity of an assertion, however, this must be done in such a way that the prover cannot convince the verifier of the validity of some false assertion.

Any interactive proof system fulfills two properties: completeness and soundness. Completeness states that if the assertion the prover wants to convince the verifier is true, then the verifier should be convinced with probability one. On the other hand, soundness is fulfilled if the verifier cannot be convinced, up to a negligible probability, of a false assertion. Therefore, completeness and soundness allow the verifier to check whether the assertion of the prover is true or false.

Zero-knowledge is a property of the prover (strategy). Consider the following informal notion of (quantum) computational zero-knowledge strategy, which corresponds to the straightforward lifting to the quantum setting of the classical version:

Definition 11.7. A prover strategy *S* is said to be *quantum computational zero-knowledge* over a set *L* if and only if for every quantum polynomial-time verifier strategy, *V* there exists quantum polynomial-time algorithm *M* such that $(S,V)(l)$ is (quantum) computationally indistinguishable from $M(l)$ for all $l \in L$, where *(S, V)* denotes the output of the interaction between *S* and *V*.

The main application of zero-knowledge proof protocols in the cryptographic setting is in the context of a user *U* that has a secret and is supposed to perform some steps, depending on the secret. The

problem is how can other users assure that *U* has carried out the correct steps without *U* disclosing its secret. Zero-knowledge proof protocols (ZKP) can be used to satisfy these conflicting requirements.

Zero-knowledge essentially embodies that the verifier cannot gain more knowledge when interacting with the prover than by running alone a quantum polynomial time program (using the same input in both cases). That is, running a the verifier in parallel with the prover should be indistinguishable of some quantum polynomial time program.

Actually, the notion of (quantum computational) zero-knowledge proofs can be captured through emulation very easily. Assuming that a proof strategy *S(x)* and verifier *V(x)* are modelled as terms of the process algebra, it is actually possible to model the interaction between *p* and *v* by the process *(p : S)* || (*v* : *V).* Denote by $L^v(l)$ the set of all process terms for the verifier $(v:V)_l^x$, that is, any process term *(v : V)* where the free variable *x* was replaced by the binary word *l*. We have the following characterisation:

Proposition 11.4. A process term (p *:* S*)* denoting a proof strategy is compulational zero- knowledge for L if and only if $(p:S)_l^x \equiv^v_{L^v(l),L^v(l)} 0$, for all $l \in L$.

Proof. Proof (Sketch): Notice that the ZKP resumes to impose that for all $(v:V)_l^x$ there is a process $(v:V')_l^x$ such that $(p:S)_l^x \parallel (v:V)_l^x \approx 0 \parallel (v:V')_l^x$. Since the semantics of a local process can be modelled by a QPM, and moreover $0 \parallel (v:V')_l^x$ can model any QPM, the characterisation proposed in this proposition is equivalent to Definition 11.7.

So, a process (p : *S*) models a quantum zero-knowledge strategy if, from the point of view of the verifier, it is impossible to distinguish the final result of the interaction with (p : *S*) from the interaction with the 0 process. A clear corollary of Theorem 11.3 is that, quantum zero-knowledge is compositional.

It is simple to adapt the emulation approach to several other quantum security properties, like quantum secure computation, authentication and so on.

Advancement questions

1. What is the role of security parameters?
2. $0 \in \acute{G}$ (global termination) ;
   $P\|$ (a : Q) $\in \acute{G}$ where $P \in \acute{G}$, $a \in A$ does not occur in P, and $Q \in L$ (global parallel composition).

What does the following example describe?

3. What we need to introduce for order in order to define the semantics of a local process?
4. What is the base of the semantcis of a local process?
5. What can be used to model the scheduler policy
6. What does global communication rule make?

2. What is the base of the notion of observational equivalence?
3. What the most successful ways for defining secure concurrent cryptographic tasks?
4. How we can called the agents of a two party protocol?
5. What the main application of zero-knowledge proof protocols in the cryptographic setting

REFERENCES

[1] Pedro Miguel dos Santos Alves Madeira Adão Formal Methods for the Analysis of Security Protocols / Pedro Miguel dos Santos Alves Madeira Adão // PhD diss., INSTITUTO SUPERIOR TÈCNICO. – 2006.

[2] Mitchell J. C. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols / J. C. Mitchell, A. Ramanathan, A. Scedrov, V. Teague //Electronic Notes in Theoretical Computer Science. - 2001. - №45. - p. 1–31.

[3] Mateus P. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus / editors R. Amadio and D. Lugiez. - Concurrency Theory. - 2003. - Vol2761. - p. 327–349.- 523 p.

[4] Cook S. A. Time bounded random access machines / S. A. Cook and R. A. Reckhow //Journal of Computer and System Sciences. - 1973. - Vol7. - p. 354–375.

[5] Knill E. Conventions for quantum pseudocode : Technical Report LAUR-96-2724 / E. Knill. - Los Alamos : National Laboratory, 1996. - 13 p.

[6] Mateus P. Reasoning about quantum systems / editors J. J. Alferes, J. A. Leite . - Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA), 2004. - Vol3229. - p.239–251. - 743p.

[7] Mateus P. Weakly complete axiomatization of exogenous quantum propositional logic / P. Mateus, A. Sernadas // Information and Computation. - 2006. - Vol204(5). - p. 771–794.

[8] Cohen-Tannoudji C. Quantum Mechanics / C. Cohen-Tannoudji, B. Diu, and F. Laloë. - John Wiley, 1977. - 887 p.

[9] Martins A. M. Minimization of quantum automata / A. M. Martins, P. Mateus, A. Sernadas // Technical report, CLC, Department of Mathematics, Instituto Superior Técnico. - 2005. - 308p.

[10] Abadi M. A calculus for cryptographic protocols: The Spi Calculus. Information and Computation / M. Abadi A. D. Gordon // SRC Research Report 149. - 1999. - Vol148(1). - p.1–70.

[11] Canetti R. Security and composition of multiparty cryptographic protocols / R. Canetti // Journal of Cryptology. - 2000. - Vol13(1). - p.143–202.

[12] Micali S. Secure computation / S. Micali, P. Rogaway // In Feigenbaum. - 2000. - p. 392–404.

CHAPTER 12. INTELLECTUAL METHODS FOR SECURITY

Content of the PART3

12.1 Application of Artificial Intelligence in Network Intrusion Detection

Introduction

Intrusion Detection System (IDS) is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusion [1,2]. It is useful not only in detecting successful intrusions, but also in monitoring attempts to break security, which provides important information for timely counter-measures. Basically, IDS can be classified into two types: Misuse Intrusion Detection and Anomaly Intrusion Detection. Traditional protection techniques such as user authentication, data encryption, avoiding programming errors, and firewalls are used as first lines of defense for computer security. If a weak password is compromised, user authentication cannot prevent unauthorized use. Also, firewalls are vulnerable to errors in configuration and susceptible to ambiguous or undefined security policies.

Recently, the use of Artificial Intelligence (AI) techniques has been employed in different data mining and machine learning classification and prediction modeling schemes. In addition to these, hybrid data mining schemes, hierarchical hybrid intelligent system models, and ensemble learning approaches that combine the base models with other hybrid machine learning paradigms, to maximize the accuracy and minimize both root mean squared errors and computational complexity, have also gained popularity in the literature [3].

In this chapter, a succinct review has been carried out on the individual capabilities of various AI techniques in their application to network IDS. Such techniques include Artificial Neural Networks (ANN), Support Vector Machines (SVM), Genetic Algorithms (GA) and Fuzzy Neural Networks (FNN). Attempts were also made to propose possible hybrid approaches based on these techniques.

Background Knowledge

Overview of Intrusion Detection Systems

Intrusion Detection Systems are used to monitor computers or networks for unauthorized entrance or activities, thereby making it easy

to detect if a system is being targeted by an attack. Preventing, detecting, and reacting to intrusions without disturbing the operations of existing systems remain a big challenge for networks that provide round the clock services such as web servers. In such networks, even if an intrusion is detected, the system cannot be shut down to check it fully since it may be serving users who are making deals or completing one transaction or the other [2].

An IDS inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system. Generally, an IDS detects unwanted manipulations of computer systems, mainly through the Internet. The manipulations may take the form of attacks by crackers [4].

An IDS is composed of three major components: Sensors which generate security events, a Console to monitor events, initiate alerts and control the sensors, and a Central Engine that records events logged by the sensors in a database and uses a system of rules to generate alerts from security events that have been received [5].

There are three major categories of IDS viz. Misuse Detection vs. Anomaly Detection, Network-Based vs. Host- Based Systems and Passive System vs. Reactive System.

Misuse Detection vs. Anomaly Detection:

The Misuse Detection part analyses the information it gathers, and compares it to large databases of attack signatures by looking for a specific attack that has already been documented while the Anomaly Detection part monitors network segments to compare their state to the normal baseline defined by the systems administrator and look for anomalies [6].

Network-Based *vs.* Host-Based Systems:

In the Network-based system, the network analyses individual packets of information flowing through it and detects those that are malicious but designed to be overlooked by a firewall's simplistic filtering rules. In a Host- based system, the IDS examines the activity on each individual computer or host [6].

Passive System vs. Reactive System:

In a Passive System, the IDS detects a potential security breach, logs the information and raises an alert signal while the reactive system

responds to the suspicious activity by logging off a user or by reprogramming the firewall to block network traffic from the suspected malicious source [6].

Overview of Artificial Intelligence

The application of the capabilities of Artificial Intelligence techniques has been widely appreciated in Computer and Communication Networks in particular, as well as in other fields. This inter-disciplinary endeavor has created a collaborative link between Computer Scientists and Network Engineers in the design, simulation and development of network intrusion models and their characteristics. Computational Intelligence (CI), an offshoot of AI, covers all branches of science and engineering that are concerned with the understanding and solving of problems for which effective computational algorithms do not yet exist. Thus, it overlaps with some areas of Artificial Intelligence and a good part of Pattern Recognition, Image Analysis and Operations Research. It is based on the assumption that thinking is nothing but symbol manipulation. Thus, it holds out the hope that computers will not merely simulate intelligence, but actually achieve it. CI relies on heuristic algorithms such as in Fuzzy Systems, Neural Networks, Support Vector Machines and Evolutionary Computation. In addition, CI also embraces techniques that use Swarm Intelligence, Fractals and Chaos Theory, Artificial Immune Systems, Wavelets, etc. [7].

AI is itself an advancement of the concept of its predecessor, Data Mining (DM). DM is the process of finding previously unknown, profitable and useful patterns embedded in data, with no prior hypothesis. It is the process of analyzing data from different perspectives, summarizing it into useful information and finding correlations or patterns among datasets in large data repositories. The objective of DM is to use the discovered patterns to help explain current behavior or to predict future outcomes. DM borrows some concepts and techniques from several long-established disciplines viz. Artificial Intelligence, Database Technology, Machine Learning and Statistics. The field of DM has, over the past couple of decades, produced a rich variety of algorithms that enable computers to learn new relationships/knowledge from large datasets [8].

AI naturally transformed into Computational Intelligence (CI) with the introduction of the concept of Machine Learning. This is a scientific aspect of AI that is concerned with the design and development of algorithms that allow computers to learn based on data, such as a network intrusion log acquired over a considerable period of time. A major focus of machine learning research is to automatically learn to recognize complex attributes and to make intelligent decisions based on the correlations among the data variables. Hence, machine learning is closely related to fields such as statistics, probability theory, data mining, pattern recognition, artificial intelligence, adaptive control, and theoretical computer science.

The machine learning concept can be categorized into three common algorithms viz. supervised, unsupervised and hybrid learning. Supervised learning is the type of machine learning technique in which the algorithm generates a function that maps inputs to the desired outputs with the least possible error. Unsupervised learning is the machine learning technique in which a set of inputs are analyzed without the target output. This is also called clustering. The hybrid learning combines the supervised and unsupervised techniques to generate an appropriate function and to meet a specific need of solving a problem. The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory [8].

A general modeling framework for computational intelligence is shown Figure 12.1.

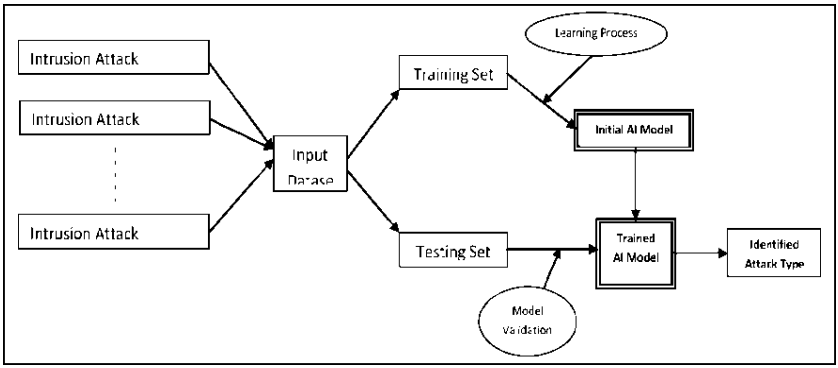


Figure 12.1. Computational Intelligence Modeling Framework

Overview of Some Artificial Intelligence Techniques and their Application in IDS

A good number of studies have been carried out on the use of various CI/AI techniques to model various IDS strategies. Some of these techniques will be discussed in the following sections.

Artificial Neural Networks (ANN)

Attempts to artificially simulate the biological processes that lead to intelligent behavior culminated in the development of ANN. ANN is a mathematical or computational model that is based on biological neural networks. It consists of an interconnected group of artificial neurons which processes information using a connectionist approach to computation. In most cases, ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.

In more practical terms, neural networks are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. A typical ANN framework is shown in figure 12.2 ANN is a close emulation of the biological nervous system. In this model, a neuron multiplies the inputs by weights, calculates the sum, and applies a threshold. The result of this computation would then be transmitted to subsequent neurons. Basically, the ANN has been generalized to:

$$y_i = f\left(\sum_k w_{ik} x_k + \mu_i\right)$$

where x_k are inputs to the neuron i, w_{lk} are weights attached to the inputs, p_t is a threshold, offset or bias, f (•) is a transfer function and y_i is the output of the neuron. The transfer function f (•) can be any of: linear, non-linear, piece-wise linear, sigmoidal, tangent hyperbolic and polynomial functions.

Some of the versions of ANN, depending on which algorithm is used at the summation stage, include: Probabilistic Neural Networks, Generalized Regression Neural Networks and Multi-Layer Perceptron Neural Networks. The most commonly used learning algorithm of ANN is the Feed-Forward Back-propagation algorithm.

More details of this technique can be found in [9, 10].

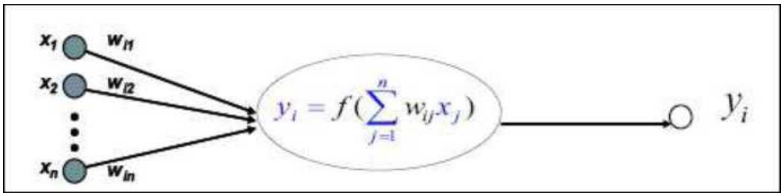


Figure 12.2. A typical Artificial Neural Networks Framework [9]

Fuzzy Inference Systems (FIS)

Fuzzy Inference System include Type-1 Fuzzy System (fuzzy) and Type-2 Fuzzy System (fuzzy fuzzy). Type-2 Fuzzy System (FS) was recently introduced as an extension of the concept of Type-1 Fuzzy. Type-2 FS have grades of membership that are themselves fuzzy. For each value of primary variable (e.g. pressure and temperature), the membership is a function (not just a point value). This is the secondary Membership Function (MF), whose domain, the primary membership, is in the interval (0,1), and whose range, secondary grades, may also be in (0,1). Hence, the MF of a Type-2 FS is three-dimensional, and the new third dimension provides new design degrees of freedom for handling uncertainties. The basic structure of a Type-2 FS is shown in Figure 3. More details of this technique can be found in [11, 12].

Sampada et al. [13] proposed two machine learning paradigms: Artificial Neural Networks and Fuzzy Inference System, for the design of an Intrusion Detection System. They used SNORT to perform real time traffic analysis and packet logging on IP network during the training phase of the system. They constructed a signature pattern database using Protocol Analysis and Neuro-Fuzzy learning method. They then tested and validated the models using the 1998 DARPA Intrusion Detection Evaluation Data and TCP dump raw data. The data set contains 24 attack types. The attacks fall into four main categories viz. Denial of Service (DOS), Remote to User (R2L), User to Root (U2R), and Probing. From the results, it was shown that the Fuzzy Inference System was faster in training, taking few seconds, than the Artificial Neural Networks which took few minutes to converge. Generally, both techniques proved to be good, but with the Fuzzy Inference System having an edge over Artificial Neural Networks with its higher classification accuracies. Their experiment also showed the importance of variable selection, as the two techniques performed worse when all the variables were used without selection of the variables. Good results were recorded when a subset (about 40%) of the variables were used.

In a similar study, [14] proposed a conceptual framework comprising of the techniques of neural networks and fuzzy logic with network profiling, using both network traffic and system audit data as inputs to the systems. The proposed system was planned to be a hybrid system that combines anomaly, misuse and host based detection. The authors planned to use neural networks with self organizing maps for host based intrusion detection. They hoped to be able to trace back suspicious intrusions to their original source path and so that the traffics from that particular source will be redirected back to them thereafter.

More studies on the application of ANN and Fuzzy Logic can be found in literature.

Support Vector Machines

Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification and regression. They belong to a family of Generalized Linear Classifiers. They can also be considered as a special case of Tikhonov Regularization. SVMs map input vectors

to a higher dimensional space where a maximal separating hyperplane is constructed [15]. This is shown in Figure 12.4.

The generalization ability of SVMs is ensured by special properties of the optimal hyperplane that maximizes the distance to training examples in a high dimensional feature space. SVMs were initially introduced for the purpose of classification until 1995 when Vapnik et al., as reported in [16], developed a new e-sensitive loss function technique that is based on statistical learning theory, and which adheres to the principle of structural risk minimization, seeking to minimize an upper bound of the generalization error. This new technique is called Support Vector Regression (SVR). It has been shown to exhibit excellent performance. Further details on SVM can be found in [17, 18, 19].

In [20], Zang and Shen utilized the capability of SVM to formulate an Intrusion Detection System as a binary classification problem by characterizing the frequencies of the system calls executed by the privileged programs. Using the intersection of pattern recognition and text categorization domains, they modified the conventional SVM, Robust SVM and one-class SVM; and compared their performances with that of the original SVM algorithm. Using the 1998 DARPA BSM data set collected at MIT's Lincoln Labs, they verified that the modified SVMs can be trained online and the results outperform the original ones with fewer Support Vectors (SVs) and less training time without decreasing detection accuracy.

Genetic Algorithms

Genetic Algorithm (GA) is a computing technique used as an exhaustive search paradigm to find exact or approximate solutions to optimization problems. GAs are categorized as global search heuristics. Its paradigm is based on a particular class of evolutionary algorithms that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. GAs are implemented in a computer simulation framework in which a population of abstract representations (representing chromosomes) of candidate solutions (representing biological creatures, or phenotypes) to an optimization problem produces better solutions. Traditionally, solutions are represented in bits (a set of 0s and 1s), but other encodings are also possible.

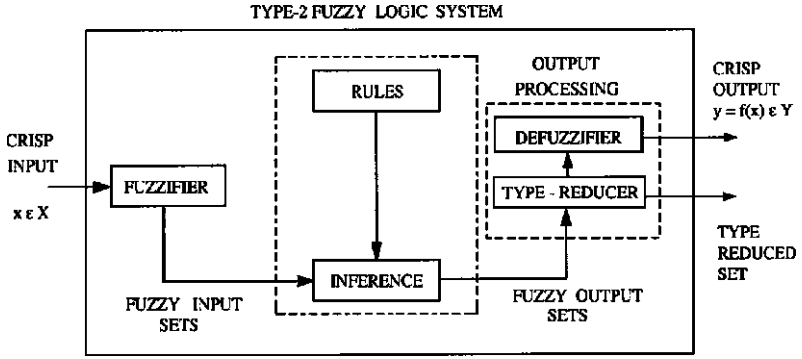


Figure 12.3. Structure of a Type-2 Fuzzy Logic System [12]

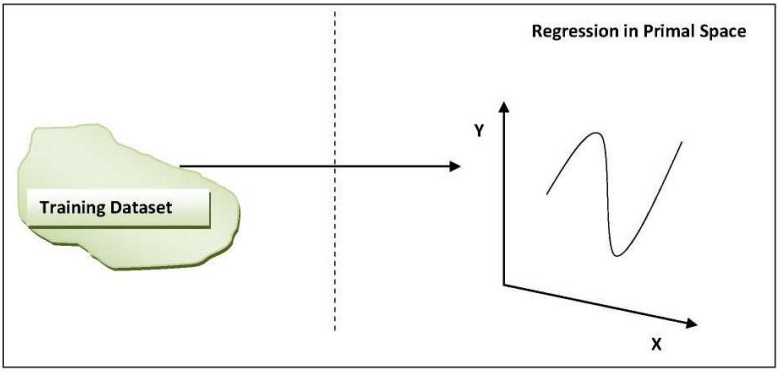


Figure 12.4. Mapping Input Vectors to a Higher Dimensional Space in SVM [23, 25, 26]

The evolution process begins with a population of randomly generated individuals and continues in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Usually, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number

of generations, a satisfactory solution may or may not have been reached.

Genetic Algorithms have been widely applied in almost all fields of research. The main property that makes genetic representations in computer simulations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. The fitness function is defined over the genetic representation and measures the quality of the represented solution. Once the genetic representation of a problem has been obtained, and the fitness function defined, GA proceeds to initialize a population of solutions randomly, and then improves it through repetitive application of mutation, crossover, inversion and selection operators.

More details on GA can be found in [21, 22].

Functional Networks

Functional Networks (FN) is an extension of Artificial Neural Networks which consists of different layers of neurons connected by links. Each computing unit or neuron performs a simple calculation: a scalar, typically monotone, function *f* of a weighted sum of inputs. The function *f*, associated with the neurons, is fixed and the weights are learned from data using some well-known algorithms such as the least-squares fitting algorithm.

The main idea of FN consists of allowing the *f* functions to be learned while suppressing the weights. In addition, the *f* functions are allowed to be multi-dimensional, though they can be equivalently replaced by functions of single variables. When there are several links, say m, going from the last layer of neurons to a given output unit, we can write the value of this output unit in several different forms (one per different link). This leads to a system of *m-1* functional equations, which can be directly written from the topology of the Neural Network. Solving this system leads to a great simplification of the initial functions f associated with the neurons.

As shown in Figure 12.5, a FN consists of a layer of input units which contains the input data, a layer of output units which contains the output data, and one or several layers of neurons or computing units which evaluates a set of input values coming from the previous layer and gives a set of output values to the next layer of neurons or output units. The computing units are connected to each other, in the sense that

output from one unit can serve as part of input to another neuron or to the units in the output layer. Once the input values are given, the output is determined by the neuron type, which can be defined by a function.

For example, assume that we have a neuron with s inputs: $(x1\ x_s)$ and k outputs: $(y1, ..., y)$, then we assume that there exist k functions F_j; $j = 1\ k$, such that $y = F_j(x1\ x_s)$; $j = 1\ k$.

FN also consists of a set of directed links that connect the input layer to the first layer of neurons, neurons of one layer to neurons of the next layer, and the last layer of neurons to the output units. Connections are represented by arrows, indicating the direction of information flow [23].

The least squares fitting algorithm has the ability to learn itself and to use the input data directly, by minimizing the sum of squared errors, in order to obtain the parameters, namely the number of neurons and the type of kernel functions, needed for training. The FN learning process consists of initial network creation, modification of the initial network, and selection of the best model. More details can be found in [25].

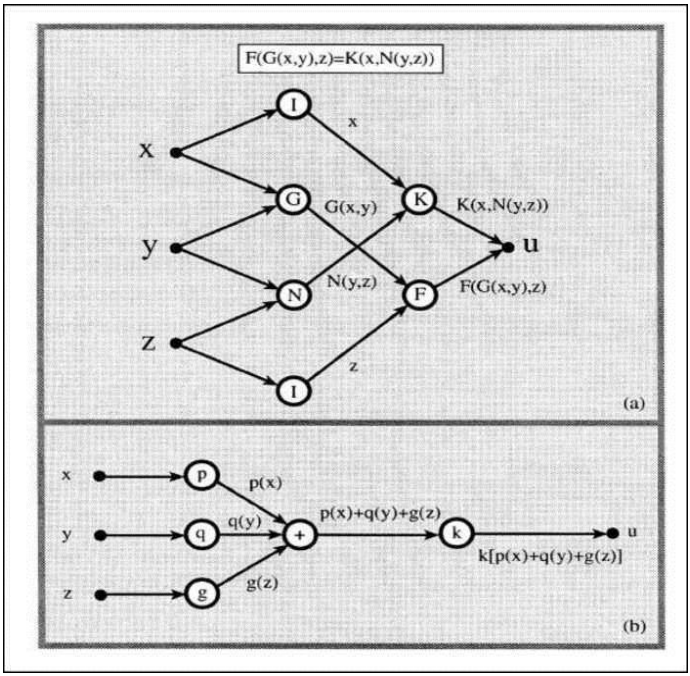


Figure 12.5. Illustration of the Generalized Associativity Functional Network. (a) Initial network (b) simplified network [24]

Advances in Artificial Intelligence Hybrid and Ensemble Techniques in IDS

Ensemble and hybrid techniques are becoming increasingly popular [25, 26]. Both methodologies have improved the performance of machine learning systems and have been successfully applied in many real world problems. The increased popularity of hybrid intelligent systems in recent times lies in their extensive success in many real-world complex problems. A key prerequisite for the merging of technologies is the existence of a "common denominator" to build upon [25]. The basic idea underlying an ensemble learning method is employing multiple learners to learn partial solutions to a given problem and then integrating these solutions to construct a final or

complete solution to the original problem [26]. However, there are several open issues in ensemble learning. For a given task, one of them is how to automatically generate an ensemble structure for taking advantage of available learners whose capabilities have been well studied or known.

Fuzzy Neural Networks with Genetic Algorithms

One of the implementations of hybrid techniques is [2] which proposed a Fuzzy Neural Network assisted with GA (FNN/GA) which used the FNN component to make a restriction of membership function to be some specific shape such as triangular, trapezoidal or bell-shaped and then tuning the parameters of the membership function with the GA component to achieve the mapping accuracy. The FNN consists of 4 layers. Layer 1 with 4 nodes consists of input and output nodes representing input and output linguistic variables respectively. Nodes in layer 2 are those that act as membership functions and each is responsible for mapping an input linguistic variable into a possibility distribution for that variable. Thus, together, all the layer 3 nodes formulate a fuzzy rule basis. Links between layer 3 and 4 function as a connectionist inference engine. The training algorithm consists of first constructing and training the FNN using the back-propagation algorithm to obtain membership functions and the consequent weight vector. The membership functions with a group of line segments that are obtained by partitioning and sampling the line segments are also constructed and finally, for every partition point, the GA is used to search the optimal value and to obtain the optimal membership functions.

Hybrid of Functional Network, Support Vector Machines and Type-2 Fuzzy Logic

Another recent implementation of hybrids is [27] which combined the excellent features of Functional Networks (FN), Support Vector Machines (SVM) and Type-2 Fuzzy Logic (T2FL). There were two versions of this hybrid: FN-Fuzzy Logic-SVM (FFS) and FN-SVM-Fuzzy Logic (FSF). In the FFS version, after the FN was used to select the most relevant variables from the input data, the best variables were passed on to the T2FL block where uncertainties were removed and the SVM block performed the training and prediction tasks. In the FSF version, the best variables from the FN block were passed through the

SVM block where they transformed to a higher dimensional space for the T2FL block to use for the training and prediction tasks. An improvement to the FFS and FSF hybrid models are presented in [28].

Fuzzy Linear Programming with Support Vector Machines

Another dimension to hybridization of AI techniques was presented by [17] who proposed a combination of Fuzzy Linear Programming (LP) with SVM to resolve the seemingly unclassifiable regions for multiclass problems. The LP-SVM was trained to define the membership functions in the directions orthogonal to the decision functions. Then by the minimum or average operation for these membership functions, a membership function for each class was defined and finally, the one-against-all and pair-wise Fuzzy LP-SVMs for some benchmark datasets were evaluated to demonstrate the superiority of the proposed Fuzzy LP-SVMs over conventional LP-SVMs.

12.2 Multi-agent based approach of botnet detection in computer systems

Introduction

The analysis of the situation of development of the malware shows dynamic growth of its quantity. The most numerous classes of malware during last 10 years are Trojans and worm-viruses which spread and penetrate into computer system (CS) for the purpose of information plunder, DDoS attacks, anonymous access to network, spy actions, spamming that represents real danger [29, 30] (Fig. 12.6).

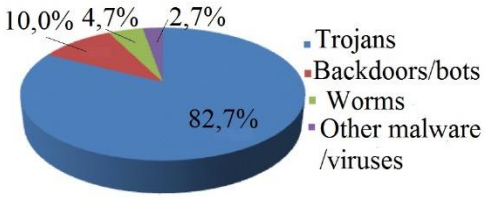


Figure 12.6: Malware rate in 2014

Despite the regular refinement of methods of the search, detecting and removal Trojans and worm-viruses of different function, regular updates of anti-virus bases, the numerous facts of plunder of the

confidential information are observed and the various destructive operations are performed which lead to serious negative consequences.

Common techniques used in modern antivirus software of Trojans' and worm-viruses' detection are signature-based one, code emulators, encryption, statistical analysis, heuristic analysis and behavioural blocking [30]. However, the accuracy of detection of new malware is low, and in recent years it has constantly decreased [31] (Fig.12.7, Fig.12.8). One of the main reasons for the lack of detection accuracy is cooperating of Trojans with worm-viruses.

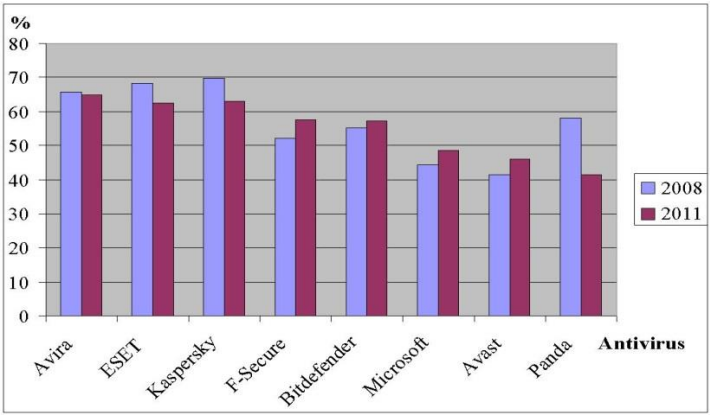


Figure 12.7: Worm-viruses' detection in 2008 vs 2011 years!!!!!!

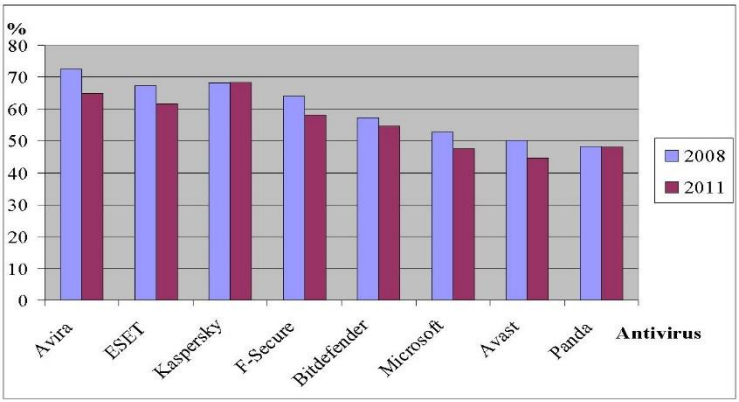


Figure 12.8: Trojans' detection in 2008 and 2011 years.

Over the past 3-5 years there is a clear dynamics of conception of a new malware class – botnet (Fig. 12.9).

Botnet today represents a real threat to computer systems users; the accuracy of its detection is low because of its complicity.

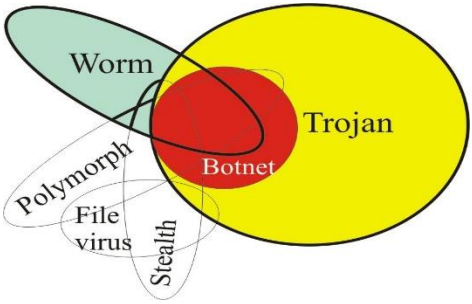


Figure 12.9: Place of Botnet among all malware

That is why the actual problem of safety of various computer systems is a development of new more perfect approach of antivirus detection. One of possible way to increase the detection efficiency is a construction of virus multi-agent system in computer system for new botnet detection. For this purpose it is necessary to develop the principles of such system functioning; to describe the communication

and functions' features of agents; to formalize sensors' and effectors' properties.

Multi-agent system of botnet detection

To increase the efficiency of botnet detection we involve multi-agent systems that will allow us to make antivirus diagnosis via agents' communication within corporate network [32].

Usage of multi-agent systems for botnet detection requires a generation of agents set with some structure and functionality [33].

Each agent should implement some behaviour and should include a set of sensors (components that directly is effected by the computer system), a set of effectors (components of that effect the computer system) and CPU - information processing unit and memory [34].

The scheme of antiviral agent multi-agent system operation is shown in Fig. 12.10.

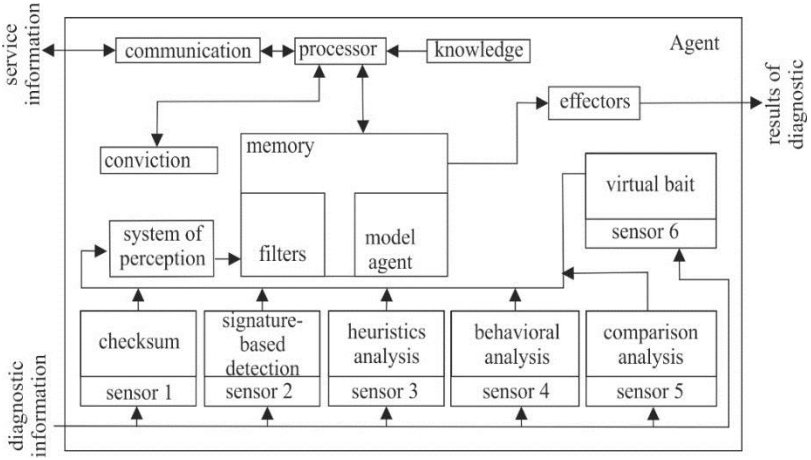


Figure 12.10: The scheme of antiviral agent multi-agent system operation.

Let us present agent as a tuple:

$$A = \langle P, R, K, S_1, S_2, S_3, S_4, S_5, S_6, .. S_n \rangle, \quad (1)$$

where P – processor, which provides integration and processing data, processing optimal response to the incoming information about the computer system state, decision on the steps to be done.

R – rules, that change agent behaviour according to incoming information.

K – agent knowledge – part of rules and knowledge, that could be changed during its functioning.

S_1 – communication sensor, communicates with other agents via network protocols.

S_2 – agent of signature-based analysis; virus detection is performed by searching signatures in database [35]; all signatures are detectors generated using the modified negative selection algorithm [36,37]; antivirus system alarms if computer is infected.

S_3 – checksum sensor.

S_4 – sensor of heuristics analysis; detection is performed in monitor mode with the use of fuzzy logic; sensor makes a conclusion about the danger degree of computer system infection with a new botnet [38].

S_5 – sensor of comparative analysis through application programming interface API and driver disk subsystem via IOS. If data on file received the first way differ from those obtained by the second way, file is infected.

S_6 – sensor - "virtual bait"; it is used for modelling of possible attacks or unauthorized access and it allows to learn the strategy of attacker and to identify a list of tools and actions intruder can do on infected computer system. If a remote administration of network is not carried out, all incoming ssh-traffic is redirected to this sensor.

The processor processes the input data and determines the level of risk of specified object in the computer system. There is a knowledge base of trusted software.

Conviction unit provides knowledge for agent in unusual situations. This will reduce the number of false positives in the new botnet diagnosis of computer system. The filters system for each sensor proposed to establish the risk factors for the evaluation of objects. Exceeding the limit values of the coefficients including the experience of all agents indicates the computer system infection with botnet.

Diagnostic information according to their functional properties each sensor is submitted. Work results of the checksums and signature analysis sensors may not require full engagement of the agent functioning for notification of the infection with botnet, but in

conjunction with results of other sensors and communication with other agents this sensors may assert this signal detection of the botnet.

Unit of perception holds summary information to the general form for further work. Then the information goes to the input of filters. Filters reject data generated by trusted programs or units (Fig.12.11).

Depending on the level of danger detected attacks the coefficients are defined by filters.

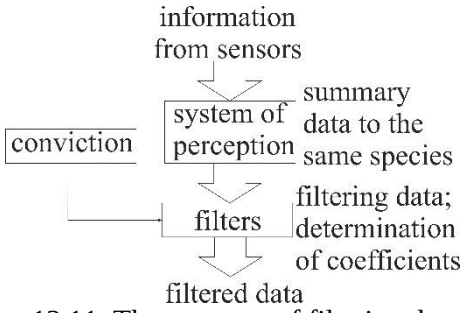


Figure 12.11: The structure of filtering data unit.

The data from the filters are to be processed by agent processor which determines whether the computer system is infected. Because of lack of data, the agent communicates with other agents for similar influence of programs' actions. The availability or absence of such information from other agents affects the final agent decision on a particular file or process.

When comparing the results obtained with the conviction unit data changes of coefficients and trusted programs are held.

Communications unit is responsible for encryption and decryption of interagents' information.

Agent results are transmitted to the effectors, as a means of influence on the computer system. If malware is detected agent through effectors blocks the process or processes that are responsible for performance of some malware and then notifies the user about the infection.

Agent model ensures the integrity of the agent's structure. It is realized by implementation of system checkpoints to provide the serviceability of this agent. Also after each checking all agent critical elements are stored for later restoring in case of virus attack on

antivirus multi-agent system or possible failures in the computer system.

Each agent can activate the recheck the selected number of sensors to refine the results.

In situation when agent cannot communicate with other agent it is as autonomous unit and is able to detect different malware relying on knowledge of the latest updates and corrections in the trusted software baseIt is advisable to keep all the given values.

Sensor of botnet detection in monitor mode

A new technique for sensor diagnosis in monitor mode which uses fuzzy logic was developed. It is based on behavioural model of malware [38]. This sensor enables to make a conclusion about the degree of danger of computer system infection by malware. For this purpose we construct the input and output linguistic variables with names: "suspicion degree of software object" - for the input linguistic variable, and "danger degree of the infection" - for output one.

The task of determination of membership function for input variable we will consider as the task of the ranking for each of mechanisms (functions) m_i of penetration ports p_j with the set of indications of danger Z and a choice of the most possible p_j with activation of some function m_i. Then we generate a matrix of advantage $M_{adv} = |q_{ij}|$. Elements of given matrix q_{ij} are positive numbers: $q_{ij} = q_i / q_j$, $0 < q_{ij} < \infty$; $q_{ji} = 1/q_{ij}$, $q_{ii} = 1$, $i, j = \overline{1,l}$, l - amount of possible results. Elements q_{ij} of matrix M_{adv} are defined by calculation of values of pair advantages to each indication separately taking into account their scales $Z = \{z_k\}$; $k = \overline{1,r}$ with usage of such formula

$$q_{ij} = \sum_{k=1}^{r} q_{ij}^{k} \cdot p_k \Big/ \sum_{k=1}^{r} q_{jk}^{k} \cdot p_k \ . \qquad (2)$$

Eigenvector $\Pi = (\pi_1, ..., \pi_m)$ is defied by using a matrix of advantage. This eigenvector answers maximum positive radical λ of

characteristic polynomial $|M_{adv} - \lambda \cdot E| = 0$. $S \cdot \Pi = \lambda \cdot \Pi$, where E is an identity matrix.

Elements of vector Π ($\sum \pi_i = 1$) are identified with an estimation of experts who consider the accepted indications of danger. The same procedure is performed for all m_i. As a result we receive a matrix of relationship $V_p = |m_i, p_j|$, in which each pair (relationship) m_i, p_j value $0 \le \pi \le 1$ responds.

Using matrix $V_p = |m_i, p_j|$, we build matrix $V_p^* = |m_i, p_j|$ in which the relationship (m_i, p_j) is used and the elements of this relationship have value π_{max} ($0 \le \pi_{max} \le 1$). Using matrix $V_p^* = |m_i, p_j|$, we build normalized curve for membership function $\mu_{X^p}(R)$ of an input variable.

Example of possible 20 pairs (x_i, y_j) ranked by the suspicion degree is given in Fig.7. Formation of function membership and at the stages of activation $\mu_{X^a}(R)$ and executing of the destructive actions $\mu_{X^e}(R)$ are similar.

As a part of the solution of the problem the FIS using Mamdani algorithm was realized (Fig. 12.12-13).

The results of fuzzy inference system 0.804 are interpreted as the degree of computer system infection with malware. If the resulting number exceeds some adopted threshold of danger antivirus system will block actions of the aqueous object. The sensor also transmits information about suspicious software to other agents.

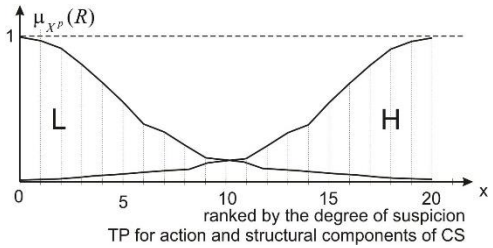


Figure 12.12: Membership function of fuzzy set “suspicion degree”

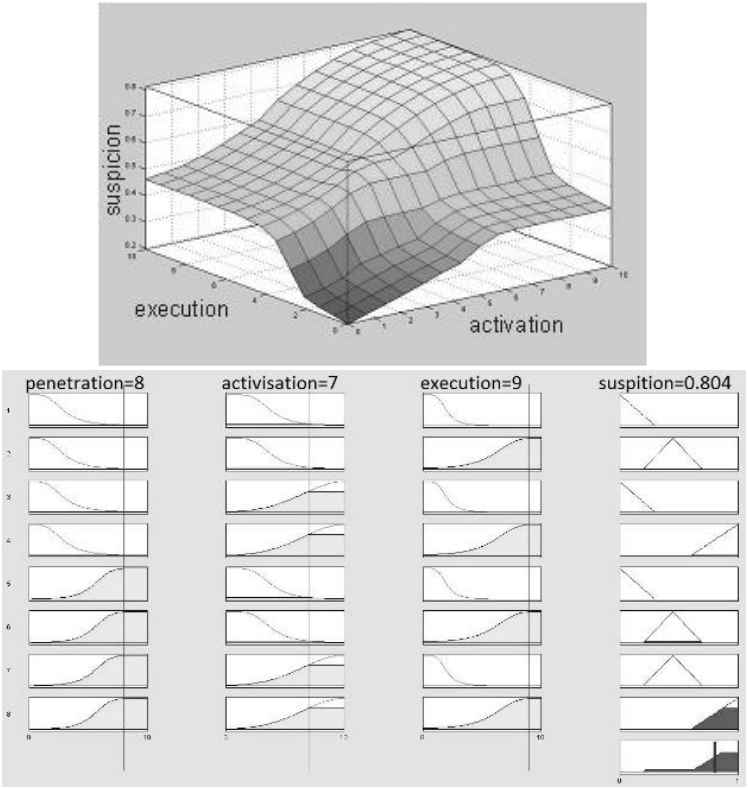


Figure 12.13: Results of the fuzzy inference system implementation and membership function of fuzzy set “suspicion degree”

Sensor of botnet detection in scanner mode

The scanner mode detection involves the following steps: forming a set of files to be scanned: system libraries, executables system services and device drivers, which can be taken as the samples; generate protected sequences and detectors depending on operating system; comparison of the protected sequences with detectors at the stage of virus scanning; notification about the substitution when the protected sequences match with detector; check the suspicion of software actions.

Thus protected sequences and detectors have format for GNU / Linux operating system:

$$D_i^L = \langle m_1..m_i..m_{x1}, u_1..u_i..u_{x2}, g_1..g_i..g_{x3}, s_1..s_i..s_{x4}, t_1..t_i..t_{x5}, C_1..C_i..C_{x6} \rangle ,(3)$$

where $m_1..m_i..m_{x1}$ - file mode (type, permissions); $u_1..u_i..u_{x2}$ - identifier of the file owner; $g_1...g_i...g_{x3}$ - identifier of the group owner; $s_1..s_i..s_{x4}$ - file size; $t_1..t_i..t_{x5}$ - time of last file modification; $C_1..C_i..C_{x6}$ - CRC of the file, $i = \overline{1,n}$, n – number of detectors.

Protected sequences and detectors have format for MS Windows operating system:

$$D_i^W = \langle s_1...s_i...s_{z1}, t_1...t_i...t_{z2}, a_1...a_i...a_{z3}, C_1...C_i...C_{z4} \rangle \qquad (4)$$

where $s_1..s_i..s_{z1}$ - file size; $t_1..t_i..t_{z2}$ - time of last file modification; $a_1..a_i..a_{z3}$ - file attribute (read-only, hidden, system, archived); $C_1..C_i..C_{z4}$ - CRC of the file, $i = \overline{1,n}$, n - number of detectors.

Generation of detectors is performed using the modified negative selection algorithm [35, 37, 38].

Agents' functioning

Let a communication agent message present as a tuple:

$$< g, h, Com_x, Com_y, Mes, t >, \qquad (5)$$

where g indicates whether it is a report, order or fetch of communication message; h - type of the agent message; Com_y -

message receiver; Com_x - message sender, Mes - agent message content; t – sending time.

Thus the communication between the units within its sensors before attack or intrusion can be represented:

$$< R, N, Se, P, Inf_{int}, t_0 > \Rightarrow < F, Int, P, M, Inf_{int}, t_1 > \frown$$
$$< O, C, P, Se, Inf_M, t_2 > \cup < O, S, P, Se, Inf_M, t_2 > \frown$$
$$< O, R, P, E, Inf_{int}, Sh, t_2 > \frown < O, I, P, Sh, Inf_{int}, t_2 >, \quad (6)$$

where R - report, O - order, F - fetch of the communication messages; N - new attack, Int - intrusion, C - continue, S - stop, Red - redirect, I - initialization as a type of the massage; P – agent processor; Se – sensors $S_1..S_5$; Sh – virtual bait; E – effectors; - are respectively the sender and receiver of the message; Inf - the content of the message; t - time of the message sending.

The communication (interactions) between the units within its sensors after attack or intrusion can be represented:

$$\text{AttackApproved} \Rightarrow < R, At, P, M, Inf_{int}, t_3 > \frown$$
$$< R, At, P, E, Inf_{int}, t_3 > \frown < O, S, P, Sh, Inf_{int}, t_3 > \frown$$
$$< R, At, P, Se, Inf_{int}, t_3 > \quad (7)$$

$$\text{AttackDisapproved} \Rightarrow < O, S, P, Sh, Inf_{int}, Se, t_3 > \frown$$
$$< O, Red, P, E, Inf_{int}, Se, t_3 > \frown < O, C, P, Se, Inf_{int}, Sh, t_3 >, \quad (8)$$

where At means – attack to computer system.

Let us formalize the function F which identifies the worth of agent A_l at time t and associates a real number to each of agents as the worth of that agent:

$$F: 2^{Ag} \times T \to R, \quad F(A_l, t) = \sum_{k=1}^{d} \frac{1}{T_a - t} N_{lk}^2 + \frac{1}{t} \frac{\Theta_{lk}}{N_{lk} + \varepsilon},$$
$$F(A_l \cup A_j, t) \geq F(A_l, t) + F(A_k, t), \quad l \neq k \quad (9)$$

where Ag - a set of agent units which are formed by combination of units with different types; Ta - the time of performing diagnosis actions; d - the number of agent components types; N_{lk} - the number of sensors of type k in agent A_l and Θ_{lk} - the sensors weight of type k within the agent no matter of their amount.

A good incentive for agents at the initial moments of reporting intrusion can be provided by sensors Se in the system in the sense that

they will form better coalitions and thus collaborate. As we can see in (6) no agent in AMAS can get more advantage by changing its actions. Also the function F does not increase by changing the agents set.

Experiments

Software for realisation of antivirus multi-agent system on proposed techniques was developed.

Interface results window of botnet diagnosing of computer system is shown in Fig. 12.14.

Figure 12.14: Software of botnet detection

For the experimental determination of the efficiency of developed software 217 programs with the botnets' properties were generated and launched on different amount of workstations (table 12.1).

Table 12.1: Accuracy of botnet detection with developed software.

Workstations (agents) / Botnets (number)	16	24	32	40
SDbot (80)	74%	75%	78%	81%
Rbot (49)	61%	63%	64%	67%
Agobot (54)	60%	60%	62%	63%
Spybot (18)	65%	68%	71%	75%

Mytob (16)	54%	57%	60%	63%
Accuracy, %	62.8%	64.6%	67.8%	69.8%

Accuracy of botnet detection of the developed software in comparison with known is shown in Fig. 12.14, it shows growth of accuracy by 3-5% in comparison with known antivirus software.

Also we performed false detection experiments and it is about 3-7%. But with the growth of agents amount false detection is reducing to 2-4%.

Conclusions

This section showed an approach for the botnet detection based on multi-agent system is proposed. A technique for sensor diagnosis in monitor mode which uses fuzzy logic is presented. The principles of communication between the agent's units before and after attack on the computer system were described. A new technique for sensor diagnosis in scanner mode with generation of detectors using the modified negative selection algorithm was illustrated.

12.3 Technique for bots detection which use polymorphic code

Introduction

Today the problem of cyber security is very important because the data protection problem is extremely relevant.

Virus detection is a very important task because the information pilfering, anonymous access to network, spy actions, spamming are observed.

The most dangerous occurrence in the virus elaboration is botnet - network of private computers infected with malicious software and controlled as a group without the owners' knowledge, e.g. to send spam [39].

Some examples of the most dangerous botnet are Virut, which have infected over 3 million unique users computer systems according to the report "Kaspersky Lab" [40]; botnet Zeus, is designed to attack servers and intercept personal data (damage for European customers is about € 36 million); botnet Kelihos, which performs abduction of passwords stored in the browser, sends spam, steals users' credentials. All mentioned botnets include the self-defense modules, which have contributed to their rapid spread and inefficient detection [41,42].

These facts indicate a lack of effectiveness of the known detection methods. That is why an important task is to build new techniques and approaches to identify botnet, which will take into account its properties and availability of hiding module.

Related works

To conceal the presence of botnet, the polymorphism technology is used.

For polymorphic malwares, the decryptor part of the virus is mutated at each infection, thanks to common obfuscation techniques: "garbage-commands" insertion, register reassignment, and instruction replacement. In order to detect such high-mutating viruses, several solutions have been developed.

Byte-level detection solutions. Current antiviral solutions use different techniques in order to detect malicious files. The techniques are: pattern-matching, emulation, dynamic behavioral detection, and various heuristics [43]. Authors focused on pattern-matching techniques (heuristics are aimed at new malware detection and are subject to a high false positive rates, while emulation may not always succeed; dynamic malware detection, while achieving good results, is out of the scope of this chapter). Because they have time and complexity constraints, the models and detection algorithms used in today's antiviral products are relatively simple. The detection algorithm consists of determining whether a given binary program is recognized by one of the viral signature. Since regular expressions are used as signature descriptions, antivirus products may use finite state automatons to perform linear-time detection.

Another emerging approach consists of using machinelearning techniques in order to detect malicious files [44]. Several models have been tested: data-mining [45], Markov chains on n-grams [46,47], Naive Bayes as well as decision trees [48]. These methods provide an automatic way to extract signature from malicious executables. But while the experiments have shown good results, the false positive and negative rates are still not negligible.

Structural and semantic models. In [49,50], graphs are used as a model for malwares. The control flow graph (CFG) of a malware is computed (when possible) and reduced. Then, subsets of this graph are

used as a signature. Detection of a malware is done by comparing a suspicious file against these sub-CFGs, and seeing if any part of the CFG of the file is equivalent (with a semantics-aware equivalence relation for [51]) to a sub-CFG in the viral database. The idea is that most of mutation engines' obfuscations do not alter the control flow graph of the malware.

CTPL is a variant of CTL [52], able to handle register-renaming obfuscation. Detection is done via model checking of API call sequence, while signatures extraction is done manually.

A promising approach was initiated by Preda et. al [53]. It consists in using the semantics of a metamorphic malware as a viral signature. None of them provides an automated process to extract this grammar for a given malware.

Background

In [54] botnet detection technique for determining the degree of presence of botnet based on multi-agent systems was proposed. Offered method was based on analyzing the bots actions demonstration in corporate area network. Technique provides propose the construction of a schematic map of connections which is formed by corresponding records in each antiviral agent of multi-agent systems for some corporate area network. All agents based on this information can perform communicative exchange data to each other. Proposed method is based on analyzing the bots actions demonstration in situations of intentional change of connection type in probably infected computer system.

During computer system (CS) is functioning the antivirus detection via sensors available in an each agent is performed. The antivirus diagnosis results are analyzed in order to define which of sensors have triggered and what suspicion degree it has produced. If triggering sensors are signature S_1 or checksum S_2 analyzers, the results R_{S1} and R_{S2} are interpreted as a 100% malware detection. In this situation, the blocking of software implementation and its subsequent removal are performed.

For situations when the sensors of heuristic S_3 and behavioral S_4 analyzers have triggered, the suspicion degrees R_{S3} and R_{S4} are

analyzed, and in the case of overcoming of the defined certain threshold n, $n \le \max(R_{S3}, R_{S4}) \le 100$, the blocking of software implementation and its subsequent removal are performed. If the specified threshold hasn't overcome the results R_{S3}, R_{S4} are analyzed whether they belong to range $m \le \max(R_{S3}, R_{S4}) < n$ in order to make the final decision about malware presence in CS. If the value is $\max(R_{S3}, R_{S4}) > m$ than the new antivirus results from sensors are expected. In all cases the antiviral agents information of infection or suspicion software behavior in computer system is must be sent out to other agents.

The important point of this approach is to research the situation where the results of antivirus diagnosis belong to range $m \le \max(R_{S3}, R_{S4}) < n$. In this case, the antiviral agent of CS asks other agents in the corporate area network about the similarity of suspicion behavior of some software that is similar to the botnet. After that the analysis of botnet demonstrations on computer systems of the corporate area network and the definition of the degree of a new botnet presence in the network was determined. The presence of botnet in the corporate area network was concluded by the fuzzy expert system that confirmed or disproved this fact.

The developed system has been demonstrating the efficiency of botnet detection at about 88-96%.

As some botnets use the technology of hiding malicious code (polymorphic code) today, mentioned multi-agent systems for botnet detection, where bots used such technology, have been tested. Test results were unexpected. It turned out that the developed system was not fully adapted to detect polymorphic code, and efficiency decreased by 7-12%. Also after retest some bots were detected, which had been previously identified and removed (all bots contained the polymorphic code).

That is why the actual problem is a development of a new botnet detection technique that will find out the polymorphic code in bots.

Technique for bots detection which use polymorphic code

In order to develop the technique we have to investigate the properties of polymorphic viruses. They create varied (though fully functional) copies of themselves as a way to avoid detection by anti-

virus software. Some polymorphic virus use different encryption schemes and require different decryption routines. Thus, the same virus may look completely different on different systems or even within different files. Other polymorphic viruses vary instruction sequences and use false commands in the attempt to thwart anti-virus software. One of the most advanced polymorphic viruses uses a mutation engine and random-number generators to change the virus code and its decryption routine [55].

Levels of polymorphism

Today 6 polymorphism levels are known [56]. Let us build models for all the levels of polymorphism.

The first level of polymorphism

Viruses of the first polymorphism level use the constant set of actions for different decryption modules. They can be detected by some areas of permanent code decryption.

Let us present the virus model for the first level of polymorphism as a tuple

$M_1 = (A, V, X, G, U, \xi, Q, P, R)$,

where $A = a_1,...,a_n$ - a set of commands of some program which can be infected with virus; V – a set of virus commands for selection of one of the present decryption modules in virus, $V = \nu_1,...,\nu_m$; X - a set of decryption modules which are present in virus, $X = x_1,...,x_y$; G - set of virus commands of the x_i decryption module, $G = \{g_{1\,xi},..,g_{\theta\,xi}\}$; U - a set of malicious commands (virus body), $U = u_1,...,u_w$; ξ - a function for selection of decryption module x_i, $\xi : V \to X$, $x_i \in X$; Q – a function of creation the malicious commands (virus body) by the means of commands $g_{x_i} \in G$ of the decryption's module x_i, $Q : G_{x_i} \to U$; P – a function of creation the polymorphic virus behavior R by the means of inserting the malicious commands U into program's commands A, $P : A \times U \to R$; function of creation the polymorphic virus behavior R without inserting malicious commands U into program's commands A

only by the means of the decryption virus body U appears as $Q: U \to R$.

Thus, polymorphic virus has its behavior that is formed by some sequences of commands. Based on this we can build the virus behavior as sequences.

Virus behavior R_1^A of the first polymorphism level which is created by the means of inserting the malicious commands U into program's commands A and virus behavior R_1 which is created without inserting the malicious commands U into program's commands A can be presented as sequences:

$R_1^A = g_{\phi_{x_\varepsilon}} .. g_{\eta_{x_\varepsilon}} a_1 .. a_n u_1 .. u_w$, $R_1 = g_{\phi_{x_\varepsilon}} .. g_{\eta_{x_\varepsilon}} u_1 .. u_w$ where values ϕ, η indicate that possible virus commands of decryption module $g_{\phi_{x_\varepsilon}} .. g_{\eta_{x_\varepsilon}}$ can vary for different decryption modules x_ε, ε - number of the selected decryption module.

The second level of polymorphism

The second level of polymorphism includes viruses, which decryption module has constant one or more instructions. For example, they may use different registers or instructions in some alternative decryption module. These viruses can also be identified by a specific signature in the decryption module [56].

Let us present the virus model for the second level of polymorphism as a tuple

$M_2 = (A, E, U, P, Z, R)$

where A - a set of commands of some program which can be infected with virus, $A = \{a_1, ..., a_n\}$; E – a set of virus commands of the decryption module, $E = (e_1 .. e_\theta)$; U - a set of malicious commands (virus body), $U = u_1, ... u_w$; Z – a function of creation the malicious commands (virus body) by the means of selection of the present decryption module's commands, $Z: E \to U$; P - a function of creation the polymorphic virus behavior R by the means of inserting malicious commands U into program's commands A, $P: A \times U \to R$; function of

creation the polymorphic virus behavior R without inserting malicious commands U into program's commands A appears as: $Z: E \times U \to R$.

Virus behaviors R_2^A and R_2 of the second polymorphism level can be presented as sequences:

$R_2^A = e_\kappa..g_\lambda a_1..a_n u_1..u_w$, $R_2 = e_\kappa..g_\lambda u_1..u_w$, where values κ, λ indicate that possible virus commands $e_\kappa..g_\lambda$ of decryption module can vary for each new start of virus.

The third/fourth levels of polymorphism

Viruses that use decryption commands and do not decrypt the virus code and have a "garbage-commands" refer to the third level of polymorphism. These viruses can be determined using a signature if all the "garbage-commands" are discarded. Viruses of the fourth level use the interchangeable or "mixed" instructions for the decryption without changing the decryption algorithm.

Let us present the virus model for the third and fourth levels of polymorphism as a tuple

$$M_{3,4} = (A, E, U, B, Y, D, R)$$

where A - a set of commands of some program which can be infected with virus, $A = \{a_1,...,a_n\}$; E – a set of virus commands of the decryption module, $E = (e_1..e_\theta)$; U - a set of malicious commands (virus body), $U = \{u_1,...,u_w\}$; B – a set of the "garbage-commands", $B = b_1,...,b_t$; Y – a function of creation the malicious commands (virus body) by the means of decryption module, that integrates the "garbage-commands" into malicious commands, $Y: E \times B \to U$; D - a function of creation the polymorphic virus behavior R by the means of virus body inserting malicious commands U into program's commands A, $D: A \times U \to R$; function of creation the polymorphic virus behavior R without inserting malicious commands U into program's commands A appears as: $Y: E \times B \to R$.

Virus behaviors R_3^A, R_4^A of the third and fourth polymorphism levels which are created by the means of decryption module, that integrates the "garbage-commands" into malicious commands and

inserts malicious commands U into program's commands A and virus behaviors R_3, R_4 without inserting malicious commands U into program's commands A can be presented as sequences:

$$R_3^A = e_1..e_\theta a_1..a_n u_1 b_\rho..u_w b_\varsigma, \qquad R_3 = e_1..e_\theta u_1 b_\rho..u_w b_\varsigma,$$

$R_4^A = e_1..e_\theta a_1..a_n u_\vartheta b_\rho..u_\sigma b_\varsigma$, $R_4 = e_1..e_\theta u_\vartheta b_\rho..u_\sigma b_\varsigma$, where values $\rho, \varsigma, \vartheta, \sigma$ indicate that possible "garbage-commands" and virus commands $u_\vartheta b_\rho..u_\sigma b_\varsigma$ can vary for each new start of virus.

The fifth level of polymorphism

The fifth level of polymorphism includes all properties of the above levels, and the decryption module may use different algorithms for decrypting the virus code.

Let us present the virus model for the fifth level of polymorphism as a tuple

$$M_5 = (A, V, B, X, G, U, \xi, H, D, R)$$

where A - a set of commands of some program which can be infected with virus, $A = \{a_1,...,a_n\}$; V – a set of commands for selection of one of the present decryption modules in virus, $V = \{v_1,...,v_m\}$; X - a set of decryption modules which are present in virus, $X = x_1,...,x_y$; G - a set of virus commands of the decryption module x_i, $G = \{g_{1xi},..,g_{\theta xi}\}$; U - a set of malicious commands (virus body), $U = u_1,...,u_w$; ξ - a function for selection of decryption module x_i, $\xi : V \to X$, $x_i \in X$,; B - a set of the "garbage-commands", $B = b_1,...,b_h$; H - a function of creation the malicious commands (virus body) by the means of selection of the present decryption module's x_i commands $g_{x_i} \in G$ and generation the order of its execution, $H : B \times G_{x_i} \to U$; D - a function of creation the polymorphic virus behavior R by the means of inserting malicious commands U program's commands A, $D : A \times U \to R$; function of creation the polymorphic virus behavior R without inserting malicious commands U into program's commands A by the means of

selection of the present decryption module's x_i commands $g_{x_i} \in G$ and generation order of its execution appears as $D: U \to R$.

Virus behaviors R_5^A and R_5 of the fifth polymorphism level can be presented as sequences:

$$R_5^A = g_{\phi_{x_\varepsilon}} .. g_{\eta_{x_\varepsilon}} a_1 .. a_n u_\vartheta b_\rho .. u_\sigma b_\varsigma, \qquad R_5 = g_{\phi_{x_\varepsilon}} .. g_{\eta_{x_\varepsilon}} u_\vartheta b_\rho .. u_\sigma b_\varsigma,$$

where values ϕ, η indicate that possible virus commands of decryption module $g_{\phi_{x_\varepsilon}} .. g_{\eta_{x_\varepsilon}}$ can vary for different decryption modules x_ε, ε - number of the selected decryption module, values $\rho, \varsigma, \vartheta, \sigma$ indicate that possible "garbage-commands" and virus commands $u_\vartheta b_\rho .. u_\sigma b_\varsigma$ can vary for each new start of virus.

The sixth level of polymorphism

Viruses of the sixth level of polymorphism consist of software units and parts that "move" within the body of the virus. These viruses are also called permutating.

Let us present the virus model for sixth level of polymorphism as a tuple

$$M_6 = (A, E, U, C, R)$$

Where A - a set of commands of some program which can be infected with virus, $A = \{a_1, ..., a_n\}$; E - a set of decryption module's commands, $E = e_1 .. e_\theta$; $U = u_1, ..., u_w$ - a set of malicious commands (virus body); C – a function of creation the polymorphic virus behavior R formed by program's commands, decryption commands and malicious commands as blocks in some order, $C: A \times E \times U \to R$; a function of creation the polymorphic virus behavior R formed only by the decryption commands and malicious command as blocks in some sequence appears as: $C: E \times U \to R$.

Virus behaviors R_6^A and R_6 of the sixth polymorphism level can be presented as sequences:

$R_6^A = a_1 e_\phi .. a_i e_\eta a_{i+1} u_\upsilon .. a_n u_\sigma$, $R_6^A = a_1 e_\phi u_\upsilon a_2 .. a_n e_\eta u_\sigma$, $R_6 = e_\phi .. e_\eta u_\upsilon .. u_\sigma$, $R_6 = e_\phi u_\upsilon .. e_\eta u_\sigma$, where values $\phi, \eta, \vartheta, \sigma$ indicate that possible virus commands of decryption module and malicious commands $e_\phi .. e_\eta u_\upsilon .. u_\sigma$ can vary for each new start of virus.

Polymorhic code detection sensor

To detect botnet that use polymorphic code, the inclusion of a new sensor S_7 for agent of the multi-agent system is proposed. This sensor must be a virtual environment that allows the emulation of execution some specific action towards the potentially malicious software. Responses to the actions allow to conclude that polymorphic code is present in it. Taking into account the properties of polymorphic viruses, sensor S_7 have to perform:

- provocative actions against probably infected file;
- restarts of the suspicious file for probably modified code detection;
- behavior analysis for modified code detection, based on the principles of known levels of polymorphism.

Provocative actions mean the identification of the polymorphic viruses' properties to create their own copies and to change their body when they are removed. This property often leads to the fact that the original virus can be found and removed, and its new copy will be invisible to antivirus.

Restarts of the suspicious software can show the possible change of the program body as a result of decryption. Detection such change is possible due to the construction of "fingerprints" of reference K and modified K' files and their subsequent comparison. "Fingerprints" K and K' are formed by a defined binary sequence $K = \alpha, \beta, \chi, \delta, \varepsilon$, where α - file name; β - file size; χ - last time of modification; δ - system attribute; ε - 128 byte code MD5.

Restarts of the suspicious software are performed by the algorithm:
Form the "fingerprint" K for file_M
for i=1 to p times do
execute file_M
Form the "fingerprint" K'
if K< >K' then sensor S_7 notifies processor of the $agent_i$ to block file_M;

Algorithm 12.1. Detection the polymorphic mutation in a suspicious file by its restarts

Sensor S_7 also provides the behavioral analyzer, which evaluates the program's actions with taking into account the models of polymorphic viruses of different levels. Based on knowledge of the polymorphic viruses' behaviors and botnet behaviors it is possible the botnet detection by comparing the known behaviors with the new ones. Identification of polymorphic code is performed with taking into account the rejection of possible "garbage commands", the permutations of commands, commands for decryptor selection, decryptor's commands etc. Behaviors are represented by sequences that are compared.

In order to perform the comparison the reference behaviors with the potentially malicious behavior, the approximate string matching algorithm, developed by Tarhio and Ukkonen [57], is used. It solves the k differences problem. Given two strings, text $\Psi = \varphi_1\varphi_2..\varphi_n$ and pattern $\Sigma = \varepsilon_1\varepsilon_2..\varepsilon_m$ and integer *k*, the task is to find the end points of all approximate occurrences of Σ in Ψ. An approximate occurrence means a substring Σ' of Ψ such that atmost *k* editing operations (insertions, deletions, changes) are needed to convert Σ' to Σ. The algorithm has scanning and checking phases. The scanning phase based on a Boyer Moore idea repeatedly applies two operations: mark and shift. Checking is done by enhanced dynamic programming algorithm. The algorithm needs time O(kn)) [58].

Based on knowledge of the possible botnet behaviors of bots there were generated 200 bots' behaviors. Taking into account the knowledge of the polymorphism levels there were generated 10 000 polymorphic behaviors. Each of them is represented by a sequence. The alphabet of

sequences is defined by a set of API-functions $\Omega = \{\omega_1 .. \omega_f\}$, which are the base for malware construction. For the experiment the behavior of three well-known bots [59] were constructed and investigated, which were "unknown" with respect to present base behaviors. These bots used three levels of polymorphism. The experimental results of the approximate string matching are presented in Table 12.1.

Table 12.1. The experimental results of the approximate string matching for different values of length R and parameter k

	Alphabet Ω	Length of sequence R	k-difference parameter	Number of found strings
P1	300	38	0	0
	300	38	2	0
	300	38	3	0
	300	38	4	1
	300	38	5	2
P2	300	93	0	0
	300	93	2	0
	300	93	3	1
	300	93	4	2
	300	93	5	7
P3	300	71	0	0
	300	71	2	1
	300	71	3	4
	300	71	4	14
	300	71	5	22

The results showed that an exact match (k=0) had not found a solution, however, when k=2, k=3 the number of solutions was sufficiently small. With increasing k the number of solutions was growing rapidly, but the search time for matches also increased. Thus, the experiments proved that for the detection of the similar suspicious behavior it was enough to lay down parameter k=4. In practice, the sensor s_7 stops the search approximate matches when it detects the first match.

Based on the concept of antivirus multi-agent system functioning, each agent is waiting for triggering of heuristic S_3 or behavioral S_4 sensors. If one of them have triggered or the fact of file unpacking has been detected then the suspicious file is placed in the sensor emulator S_7. Algorithm of seansor S_7 functioning is shown below.

for i=1 to k agents do
while $agent_i$ is_on do
if ($R_{S3} = true$ or $R_{S4} = true$) and ($m \le \max(R_{S3}, R_{S4}) < n$)
then probably infected file_M is placed into sensor S_7
if file_M makes unpacking
then file_M is blocked and is placed in sensor S_7
while file_M is in sensor S_7 do
if provocative actions regarding to file_M have detected the new file creation or new file creation with mutation
then sensor S_7 notifies processor of the $agent_i$ to block file_M; collected information about file_M is sent to other agents
If restarts have detected the file_M body mutation
then sensor S_7 notifies processor of the $agent_i$ to block file_M; collected information about file_M is sent to other agents
else behavior analysis is being performed
if result of behavior analysis R_{S7}=true
then sensor S_7 notifies processor of the $agent_i$ to block file_M; collected information about file_M is sent to other agents
else file_M leaves the sensor S_7

Algorithm 12.2. Sensor S_7 functioning algorithm

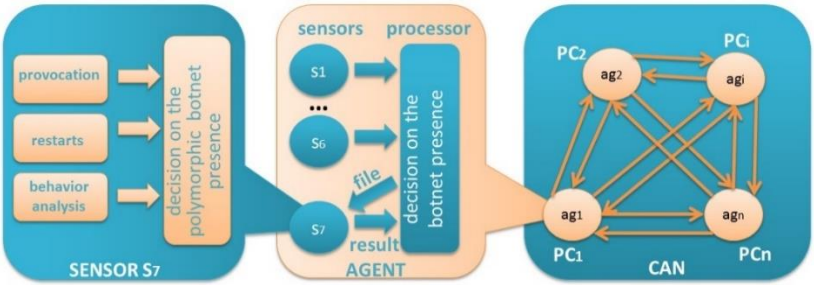


Figure 12.15 Sensor S_7 functioning in agent of the multi-agent system

Experiments

In order to determine the efficiency of the proposed technique for botnet detection several experiments were held. Bots used polymorphic code. Experiments were carried out on the base of developed multi-agent system that is functioning in the corporate area network. The main aim of the experiment was to determine the effectiveness of the botnet detection with the use of sensor S_7 and without it.

For the implementation of an experiment 50 programs with the botnet properties (Agobot, SDBot та GT-Bot) without polymorphic code were generated. Also 50 programs (its analogs) with polymorphic code were generated (programs contained only first four levels of polymorphism). During the experiment computer systems in the corporate area network were infected only by one botnet and experiment was lasting during 24 hours. As a virtual environment for sensor S_7 functioning the virtual machine Oracle VirtualBox [60] was used; as a host operating system MS Windows 7 was used.

The results of the experiment are shown in table 12.2.

Table 12.2 The results of the experiment for 50 programs

	Detection		Fault positives
	%	number	number
Results of detection without sensor s_7; Programs do not use polymorphic code	90	45	5

Results of detection without sensor s_7; Programs use polymorphic code	76	38	5
Results of detection with sensor s_7; Programs use polymorphic code	92	46	6

Experimental results showed the growth of the botnet detection efficiency which bots used polymorphic code by means of the multi-agent system including sensor S_7.

In order to compare developed antiviral multi-agent system (AMAS) with other antiviruses some experiments were held. We have tested 5 antiviruses with 50 generated bots, which contained polymorphic code. Results are presented in Fig. 12.16.

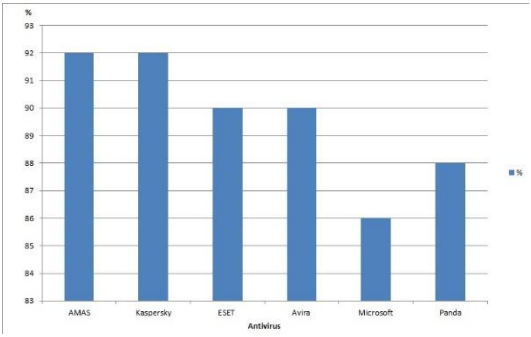


Figure 12.16 Test results (14-24.12.2013)

Conclusions

This section demonstrates the technique for botnet detection where bots use polymorphic code. Performed detection is based on the multi-agent system by means of antiviral agents that contain sensors. For detection of botnet, which bots use polymorphic code, the levels of polymorphism were researched and its models were presented.

Developed sensor performs provocative actions against probably infected file, restarts of the suspicious file for probably modified code detection, behavior analysis for modified code detection, based on the principles of known levels of polymorphism.

Results of the experiments have demonstrated the increase of the botnet detection efficiency by 16% with involving the sensor S_7 compared to its absence. Thus the growth of false positives is not significant.

The disadvantage of the proposed technique is sufficiently large computational complexity of the behavior analysis that is based on the principles of polymorphism levels.

Advancement questions

11. What kind of process is called Intrusion Detection System (IDS) and what is its purpose?
12. What are the three major components and categories of an IDS?
13. Name general characteristics of the basic categories of IDS.
14. What is the difference between AI (Artificial Intelligence) and CI (Computer Intelligence)?
15. What are the three common algorithms of the Machine Learning concept?
16. What is the Artificial Neural Networks technique and how they could be applied in IDS?
17. What are the main dificaties of the polymorphic code detectin?
18. Name all existing levels of polymorphism and how they can be detected?
19. Explain the idea of polymorphic code detection using sensor approach.
20. What is the virtual the emulation of execution some specific action towards the potentially malicious software for?

REFERENCE

1. Fatai Adesina Anifowose Safiriyu Ibiyemi Eludiora Application of Artificial Intelligence in Network Intrusion Detection / Fatai

Adesina Anifowose Safiriyu Ibiyemi Eludiora // World Applied Programming.- 2012. - Vol (2), No (3). – pp. 158-166.

2. Lee C.H. International Conference on Fuzzy Systems / C.H. Lee, Y. C. Lin // Hybrid Learning Algorithm for Neuro-Fuzzy Systems", Proceedings. 2004 IEEE - 2004. -p691-696.
3. Artail H. A Hybrid Honeypot Framework for Improving Intrusion Detection Systems Organizational Networks / H. Artail, H. Safa, M. Sraj, I. Kuwatly, Z. Al-Masri // Journal of Computers & Security.- 2006. - Vol. 25 -p274 – 288.
4. Eduardo J., and Brandão M.S. A New Approach for IDS Composition / J. Eduardo, M.S. Brandão // IEEE International Conference on Communications. - 2006.-p2195-2200.
5. Watkins A. An Immunological Approach to Intrusion Detection / A. Watkins // 12th Annual Canadian Information Technology Security Symposium. , 2000.-p447-454.
6. Intrusion Detection System[Electronic resource]: Wikipedia, The Free Encyclopedia, "Intrusion Detection System", Access mode : http://en.wikipedia.org/wiki/Intrusion-detection_system.
7. Jun H. Computational Intelligence [Electronic resource] / H. Jun // Research Interests - Access mode : http://www.cs.bham.ac.uk/~jxh/hejunrs.html.
8. Symeonidis, A. L. Agent Intelligence through Data Mining / A. L. Symeonidis // Multi-agent Systems, Artificial Societies, and Simulated Organizations Series 14. - 2000.-p. 200.
9. Petrus J.B. Artificial Neural Networks: An Introduction to ANN Theory and Practice / J.B Petrus // Springer. - 1995.-p37-57.
10. Wang Y. Fuzzy Clustering Analysis by using Genetic Algorithm // Y. Wang //, Innovative Computing, Information and Control Express Letters 2(4). - 2008.-p.331-337.
11. Castillo O. Intelligent Systems with Interval Type-2 Fuzzy Logic /O. Castillo, P. Melin // International Journal of

Innovative Computing, Information and Control 4 (4). - 2008.-p.771-784.

12. Mendel J. Type-2 Fuzzy Sets: Some Questions and Answers / J. Mendel. // IEEE Connections, Newsletter of the IEEE Neural Networks Society 1. - 2003.-p.10-13.
13. Sampada C. Adaptive Neuro-Fuzzy Intrusion Detection Systems / C. Sampada, S. Khusbu, D. Neha, M. Sanghamitra, A. Abraham, S. Sugata // International Conference on Information Technology: Coding and Computing (ITCC'04), DOI: 0-7695-2108-8/04, 2004.
14. Bashah N. Hybrid Intelligent Intrusion Detection System / N. Bashah, I.B. Shanmugam, A.M. Ahmed // World Academy of Science, Engineering and Technology. - 2005.-p.23-26.
15. Burges C.J. A Tutorial on Support Vector Machines for Pattern Recognition / C.J. Burges. // Data Mining and Knowledge Discovery 2. - 1998 -pp121-167.
16. Cristianini N. An Introduction to Support Vector Machines and other Kernel-Based Learning Methods, 1st Edition / N. Cristianini, and J. Shawe-Taylor. - Cambridge University Press, UK, 2000.
17. Abe S. "Fuzzy LP-SVMs for Multiclass Problems" / S. Abe // In Proceedings: European Symposium on Artificial Neural Networks, Belgium -2004. - p429-434.
18. Taboada J. "Creating a Quality Map of a Slate Deposit using Support Vector Machines" / J. Taboada, J.M. Matías, C. Ordóñez, and P.J. García // Elsevier Journal of Computational and Applied Mathematics 20 (4). -2007. -p84-94.
19. Xing Y. "Multiclass Least Squares Auto-Correlation Wavelet Support Vector Machines" / Y. Xing, X. Wu, and Z. Xu // International Journal of Innovative Computing, Information and Control Express Letters 2 (4). -2008. -p345-350.

20. Zhang Z. "Application of Online Training SVMs for Real-time Intrusion Detection with Different Considerations" // Z. Zhang, and H. Shen // Elsevier Journal of Computer Communications, Volume 28. -2005. -p1428-1442.
21. Mohsen S. Design of Neural Networks using Genetic Algorithm for the Permeability Estimation of the Reservoir / S. Mohsen, A. Morteza, and Y.V. Ali // Journal of Petroleum Science and Engineering, Vol. 59. -2007. – p. 97–105.
22. Bies R.R. "A Genetic Algorithm-Based Hybrid Machine Learning Approach to Model Selection" / R.R. Bies, M.F. Muldoon, B.G. Pollock, S. Manuck, G. Smith, M.E. Sale // Journal of Pharmacokinetics and Pharmacodynamics, Vol. 33. -2006. P. 195-221.
23. Castillo E. "Functional Networks" / E. Castillo // Neural Processing Letters. -1998 -Vol7. -p151–159.
24. El-Sebakhy E.A., "Software reliability identification using functional networks: A comparative study" / E. A. El-Sebakhy // Expert Systems with Applications, Volume 36. -2009. – p. 413-420.
25. Anifowose F. “Hybrid AI Models for the Characterization of Oil and Gas Reservoirs: Concept, Design and Implementation” / F.Anifowose. -VDM Verlag, 2009.
26. Inoue H. “Efficient Pruning Method for Ensemble Self-Generating Neural Networks” / H. Inoue, and H. Narihisa // Journal of Systemic,Cybernetics and Informatics. -2003. -p423-428.
27. Helmy T. “Hybrid Computational Models for the Characterization of Oil and Gas Reservoirs” / T. Helmy, F. Anifowose and K. Faisal // Elsevier International Journal of Expert Systems with Applications, vol. 37. -2010. -p5353-5363.
28. Anifowose F. “Fuzzy Logic-Driven and SVM-Driven Hybrid Computational Intelligence Models Applied to Oil and Gas

Reservoir Characterization" / F. Anifowose and A. Abdulraheem // Journal of Natural Gas Science and Engineering, Volume 3. -2011. -p505-517.

29. Goshko, S. Encyclopedia of protection against viruses / S. Goshko. - SOLON-Pres, 2005.
30. Savenko O. Research of the antivitus technologies for malware detection / O. Savenko, S. Lysenko, A Kryshchuk // Proceedings of the XII conference "Modern informations & electronic technologies - 2011", Vol1 – Ukraine, Odessaa, 2011. – p.95-96.
31. AV Comparatives laboratories [electronic resource] – Access mode http://www.av-comparatives.org. – Home page name.
32. Wooldridge M. An Introduction To Multiagent Systems / M. Wooldridge. - John Wiley & Sons LTD. - 2002. - 365 p.
33. Shoham Y. Multiagent Systems Algorithmic, Game-Theoretic, and Logical Foundations / Y. Shoham, K. Leyton-Brown // Cambridge University Press. - 2009. - 552 p.
34. Alkhateeb F. Multi-Agent Systems – Modeling, Control, Programming, Simulations and Applications / F. Alkhateeb, E. Maghayreh, I. Doush, // InTech.. - 2011. - 532 p.
35. Савенко О. Розробка процесу виявлення троянських програм на основі використання штучних імунних систем / Олег Савенко, Сергій Лисенко //Вісник Хмельницького національного університету. – 2008. – №5, – С.183-188.
36. Forrest A. Serf-nonself discrimination in a computer / A. Forrest, , L. Perelson, , R. Cherukuri // Proceedings of the IEEE Symposium on Research in Security and Privacy. - 1996. - 13p.
37. Castro L. Artificial Immune Systems: A New Computational Approach / L. Castro , T. Timmis. - London. :Springer-Verlag, 2001.
38. Bernikov A.R. Malware search in the distributed simulators using the technology of fuzzy logic / A.R. Bernikov, R.P.

Grafov, S.M. Lysenko, O.S. Savenko // Information technologies. - Moscow : 2011, № 10. - P.42-47.

39. Oxford Dictionaries http://www.oxforddictionaries.com/ definition/english/botnet?q=botnet.
40. Nikitina T. У Virut отобрали ключевые домены (2013), http://www.securelist.com/ru/blog/ 207764413/ U_Virut_otobrali_ klyuchevye_domeny#page_top (in Russian).
41. Yaneza J.: ZeuS/ZBOT Malware Shapes Up in 2013 (2013), http://blog.trendmicro.com/trendlabs-security-intelligence/zeuszbot-malware-shapes-up-in-2013.
42. Michael E. Scott: Boston Marathon/West, Texas Spam Campaigns (2013), http://mrpdchief.blogspot.com/2013/ 04/boston-marathonwest-texas-spam-campaigns.html.
43. Szor P.: The Art of Computer Virus Research and Defense. Addison-Wesley Professional (2005).
44. Kolter J. Z. Learning to detect malicious executables in the wild. / J.Z. Kolter, M.A. Maloof // In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ser. KDD '04. -2004. –p. 470–478.
45. Ye Y. Imds: intelligent malware detection system. / Y. Ye, D. Wang, T. Li, D.Ye // Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ser. KDD '07. -2007. –p. 1043–1047.
46. Griffin K. Automatic generation of string signatures for malware detection. / K. Griffin, S. Schneider, X. Hu, T.Chiueh // Lecture Notes in Computer Science, Springer Berlin. - vol. 5758. -2009. –p. 101–120.
47. Yan W. Toward automatic discovery of malware signature for anti-virus cloud computing / W. Yan, E. Wu // Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. -2009. - vol. 4. -p724–728.

48. Wang J.-H. Virus detection using data mining techinques. / J.-H. Wang , P. Deng, Y.-S. Jaw, Y.-C. Liu. // Proceedings of the IEEE 37th Annual 2003 International Carnahan Conference on Security Technology. -2003.
49. Christodorescu M. Static analysis of executables to detect malicious patterns. / M. Christodorescu, S. Jha // In: Proceedings of the 12th USENIX Security Symposium. -2003. – p. 169–186.
50. Bonfante G. Control flow graphs as malware signatures /G. Bonfante, M. Kaczmarek, J.-Y. Marion // International Workshop on the Theory of Computer Viruses TCV'07, Eric Filiol, Jean-Yves Marion, and Guillaume Bonfante, Eds., Nancy France. -2007.
51. Clarke E. Design and synthesis of synchronization skeletons using branching time temporal logic. / E. Clarke, E. Emerson // Logics of Programs, ser. Lecture Notes in Computer Science, D. Kozen, Ed. Springer Berlin. -1982. - vol. 131. – p. 52–71.
52. Leder F.Classification and detection of metamorphic malware using value set analysis. / F. Leder, B. Steinbock, P. Martini // Malicious and Unwanted Software (MALWARE), 2009 4th International. - 2009. – p. 39 – 46.
53. Preda M. A semanticsbased approach to malware detection. / M.Preda, D. Christodorescu, M. Jha, S, Debray // ACM Trans. Program. Lang. Syst., -2008. - vol. 30, no. 5. – p. 1–54.
54. Pomorova O. Multi-agent Based Approach for Botnet Detection in a Corporate Area Network Using Fuzzy Logic. / O. Pomorova, O. Savenko , S. Lysenko, A. Kryshchuk // Kwiecien A., Gaj, P., Stera, P. (eds.) CN2013. CCIS, , Springer, Heidelberg Dordrecht London New York. -2013. - vol. 370. - p.146-156.
55. Glossary. ttp://home.mcafee.com/virusinfo/glossary?ctst=1#P (2016).

56. Kaspersky E. Computer viruses / Kaspersky E. – Moscow : SK-Press, 1998.
57. Jokinen, P., Tarhio, J., Ukkonen, E.: A Comparison of Approximate String Matching Algorithms. Software: Practice and Experience 26(12), 1439-1458. http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1097-024X(199612)26:12<1439::AID-SPE71>3.0.CO;2-1/abstract (1996)
58. Smyth, B.: Computing Patterns In Strings, p. 496. Williams, Moscow (2006)
59. http://security.ludost.net/exploits/index.php?dir=bots.
60. https://www.virtualbox.org

CHAPTER 13. METHODS AND TECHNIQUES FOR FORMAL DEVELOPMENT AND QUANTITATIVE ASSESSMENT. RESILIENT SYSTEMS

Content of the chapter 13

Background: Concepts

In this chapter, we give an overview of the main phenomena concepts and properties appearing in the development of resilient distributed systems. We consider the notion of "resilience" as an evolution of the dependability concept and discuss how goal-oriented development facilitates engineering of resilient systems. In particular, we focus on the dynamic system reconfiguration as the main mechanism for achieving system resilience.

Resilience Concept

Resilience is a fairly new concept that has been intensively discussed over the last years. Though various interpretations exist, we rely on the *dependability-based* definition that was proposed by Laprie [1,2,3]:

System resilience is used to designate an ability of the system to persistently deliver its services in a dependable way even when facing changes.

Resilience is an evolution of the dependability concept that focuses on studying the impact of *changes* on system trustworthiness. A *change* is a broad term that may be viewed differently in various domains. The changes can be systematized according to their nature, prospect and timing issues [3]:

- nature: functional, environmental or technological;
- prospect: foreseen, foreseeable, unforeseen (or drastic) changes;
- timing: short term (e.g., seconds to hours), medium term (e.g., hours to months) and long term changes (e.g., months to years).

Resilience extends the dependability concept by emphasizing the need to build systems that are flexible and adaptive. It requires implementation of the advanced reconfiguration mechanisms and flexible strategies for efficient utilization of the system components to cope with changes and tolerate faults [4].

Since resilience is an evolution of the notion of dependability, majority of its concepts are grounded in the classical definitions proposed for dependability that are discussed next.

Dependability: Basic Definitions

Dependability is one of the main requirements that we impose on a broad range of computer-based systems. It can be defined as the ability of a system to deliver services that can be justifiably trusted [5, 6]. Dependability is an integrated concept that includes such key attributes as:

- *availability:* the ability of the system to provide a service at any given instant of time;
- *reliability:* the ability of the system to provide a service over a specified interval of time;
- *safety:* the ability of a system to deliver a service under given conditions without catastrophic consequences to the user(s) and environment;
- *integrity:* the absence of improper system alterations;
- *maintainability:* the ability of a system to be restored to a state in which it can deliver correct service;
- *confidentiality:* the absence of unauthorized disclosure of information.

Different *threats* may introduce undesirable deviations in service provisioning and thus jeopardise dependability. Traditionally, threats can be classified into the following categories: *failures*, *errors* and *faults* [5, 6]. Essentially, these terms designate the chain of propagation of a fault to the system boundary as defined below:

- *a failure:* an event that occurs then the delivered service deviates from the desirable (correct) service;
- *a error:* an internal system state that may lead to the subsequent system failure;
- *a fault:* a defect within the system. By their nature, faults can be internal (e.g., a software bug, a memory bit "stuck") or external (e.g., a production defect, a human mistake, an electromagnetic perturbation). In general, a fault might be an origin of an error. However, not all faults produce errors.

Traditionally, engineering dependable systems relies on four main techniques: *fault prevention*, *fault removal*, *fault forecasting* and *fault tolerance* [5, 6].

Fault prevention is a set of techniques aimed at preventing an introduction of faults during the system development process. It comprises, among others, a choice of rigorous development methodologies as well as adopting a suitable standard of quality. *Fault removal* techniques are used to identify and remove errors in the system. The activities of fault removal process include system verification as well as corrective and preventive maintenance of the system. *Fault forecasting* methods are based on prediction and evaluation of the impact of faults on the system behavior. The evaluation might have both qualitative and quantitative aspects. The qualitative assessment helps to identify and classify failures as well as define combinations of component faults that may lead to a system failure. The quantitative analysis is performed to assess the degree of satisfaction for the required attributes of dependability. Finally, *fault tolerance* techniques are used to develop the system in a such way that it is able to continue its functioning despite the faults.

All these techniques provide the designers with different means to cope with faults. The techniques complement each other and allow the designers to ensure a high degree of system dependability. The fault prevention is implemented via formal modelling, fault removal employs theorem proving to verify various system properties, while probabilistic model checking and discrete event simulation are used for fault forecasting. Moreover, we extensively rely on a variety of fault tolerance mechanisms in the development of resilient systems. Since faults constitute the most common class of changes with which a resilient system should cope, next we give an overview of the fault tolerance concepts in more detail.

Fault Tolerance. Fault tolerance techniques aim at ensuring that the system continues to deliver the required service even in the presence of faults. Fault tolerance is usually implemented in two main steps - *error detection* and *system recovery. Error detection* is used to identify the presence of errors. In its turn, *system recovery* aims at eliminating the detected errors (via *error recovery*) and preventing faults from re-activation (via *fault handling*) [7].

Error recovery takes one of the following three forms:

- *backward recovery:* bringing the system back to a previous (correct) state;
- *forward recovery:* moving the system into a new state, from which it can operate (sometimes, in a degraded operational mode);
- • *compensation:* putting the system into an error-free state (which relies on the condition that the system has enough redundancy to mask the detected error without service degrading).

In its turn, *fault handling* is a process aimed at preventing faults from being activated again. It can be conducted in three steps. The first step - *fault diagnosis* - determines the causes of errors. The next step is *isolation.* It comprises the actions required to prevent the faulty component(s) from being invoked in the further executions. The last step is *system reconfiguration.* It consists of modifying the (part of the) system structure in such a way that the system continues to provide an acceptable, but possibly degraded, service.

Fault tolerance is achieved by the reliance on redundancy. Different forms of redundancy allow the system either to mask a failure, i.e., nullify its effect at the system level, or to detect a fault and provide (usually temporary) degraded services in the presence of failures. While redundancy enables fault tolerance, it also increases complexity of the system.

Traditionally, the fault tolerance techniques are applied to cope with a number of anticipated situations including failures of software components as well as other abnormal system states. It is desirable to ensure that a system under construction reacts predictably in the presence of such abnormal situations. We can demonstrate that this task can be greatly facilitated by formal modelling.

Faults can be considered as a simple form of changes, and hence, fault tolerance constitutes an essential mechanism of achieving resilience. Fault tolerance ensures that faults do not prevent the system from delivering its services, i.e., allows it to achieve its *goals.* Since goals provide us with a suitable mechanism for representing the behavior of a complex resilient system, next we give a detailed overview of this concept.

Goal-Based Development

Goals are the functional and non-functional objectives of a system [8, 9]. In software engineering, goals have been recognized as useful primitives for capturing system requirements. The reasoning in terms of goals promotes structuring the top-down system design. Goals allow the designers to explore different architectural alternatives. They constitute suitable basics for reasoning about the system behavior. In particular, resilience can be seen as a property that allows the system to progress towards achieving its goals.

Usually, the system has different types of goals. They are often interdependent. Goals can be structured, e.g., to form a hierarchy. Generally, they can be formulated at different levels of abstraction: high-level goals represent the overall system objectives, while lower-level goals might define the objectives to be achieved by subsystems or components [8, 9]. Links between goals represent various interdependencies, i.e., the situations where goals affect each other. Traditionally, AND/OR decomposition-abstraction links are introduced to represent the intended goal structure. The process of goal detailisation (i.e., decomposition into subgoals) is performed until a certain level of granularity is reached, i.e., when a subgoal can be assigned to and consecutively realized by the system components - agents [8, 9]. Agent is an active component that performs a task and contributes to goal achievement [9, 8, 10].

The agent concept provides us with a powerful and expressive abstraction for handling complexity of distributed system development. Definition of the term agent varies across the software engineering field [11]. In this case, an agent designates a software component that is associated with a certain functionality and is capable to act autonomously in order to meet the design objectives [12, 13]. Correspondingly, *multi-agent systems* are typically decentralized distributed systems composed of agents asynchronously communicating with each other [14]. We can consider a decentralized agent system to be a system that operates without a control of central authority.

Typically, agents *interact* with each other in order to achieve their individual or common goals. Interactions might be simple, e.g., information exchange, or complex, e.g., involving requests for service provisioning from one agent to another [13].

Interactions enable agent cooperation. The level and type of cooperation between particular agents is defined by a system organization. Traditionally, we distinguish between three types of organizations: hierarchical organization (i.e., one agent may be the manager of the other agents), flat organization (i.e., agent may work together in a team and communicate with each other directly) and hybrid organization [13].

Agent *interactions* are achieved by communication. Communication allows the individual agents to share their local information with others agents to facilitate goal achievement. Traditionally, the employed communication mechanism is defined by a certain protocol describing the rules of agent interactions.

The aim of this case study we study resilience of multi-agent systems. Therefore, we should explicitly represent off-nominal situations such as agent failures or agent disconnections and assess their impact on the system behavior. As a result of these off-nominal conditions, agents usually lose an ability to perform their predefined tasks. These might prevent the system from achieving its goals and jeopardize such essential property as safety. The system should recognize such situations and autonomously reconfigure itself to prevent possible harm. Next we give an overview of the aspect of autonomous reconfiguration in detail.

System Autonomy and Reconfiguration

The concept of system autonomy has been introduced to designate systems that are able to manage themselves [15] without human intervention. Removing humans from the control has been motivated by such reasons as unfeasibility or danger of direct human involvement (due to remote or dangerous environment), a possibility to increase system performance (software usually reacts much quicker than humans) or decrease of system costs [16, 17]. The original concept of system autonomy included such “self” mechanisms as self-

configuration, self-repairing, self-healing, self-protection [15, 18]. However, nowadays the autonomic computing paradigm has been broadened and generalized.

System autonomy can be considered from different perspectives, including autonomy of the individual elements forming the system and autonomy of the whole system in general. The autonomic computing paradigm has been widely adopted in various applications and with different degree of autonomy ranging from semi-controlled by humans systems to fully autonomous [16]. Autonomous systems are currently being deployed in many critical applications such as robotics, intelligent monitoring (e.g., healthcare monitoring, traffic jam monitoring), autonomous road vehicles ("driverless cars"), etc.

Typically, the autonomic aspect assumes that a system is capable to monitor its behavior and dynamically adjust it, if needed. From the resilience perspective, system autonomy can be achieved via dynamic adaptation to various changes and volatile operating conditions. Often adaptation is performed by taking actions that transfer a system from one configuration to another. In general, the adaptation can take a form of parameter adaptation or structural adaptation. The parameter adaptation means changing the measurable system characteristics. The structural adaptation is typically performed via *dynamic reconfiguration.* Essentially, a system *configuration* can be viewed as a specific arrangement of the elements (components) that constitute the system. A configuration is defined by relationships and dependencies between system elements that are established to support achieving system goals. *Dynamic reconfiguration* in its turn assumes that the system is capable to evolve from it current configuration to another one. Dynamic system reconfiguration may imply removal or replacement of configurable elements, which consequently leads to changing of interdependencies between the components. Moreover, reconfiguration may also affect component interactions. The aim of reconfiguration is to ensure that the system remains operational, i.e., capable of achieving its goals and maintaining safe and correct delivery of its services.

We study the reconfiguration aspects in the goal-oriented and service-oriented development paradigms. In particular, the purpose of

reconfiguration is to ensure that the system goals remain achievable. The reconfiguration is based on reallocation of responsibilities between agents either to ensure that the healthy agents can substitute the failed ones (thereby, we ensure handling of negative changes) or to enable more efficient utilization of agents (hence, we address positive changes). Within service-oriented framework the reconfiguration aims at ensuring that a service can be delivered despite failures of some service-providing components. Reconfiguration here aims at utilizing the available service components to re-execute a failed service.

To effectively adapt to changes in the system and its environment, we need also to assess various reconfigurable strategies and architectural alternatives. Indeed, they can guarantee different resilience characteristics, e.g., expressed in the form of performance/reliability ratio. Hence, while developing a resilient distributed system, it is important to consider not only qualitative aspects of system resilience but also its quantitative characteristics. In general, the desirable properties and characteristics to be assessed are identified according to the system goals. In this section, we focus on the *design-time* assessment of resilience properties.

Obviously, the design and verification of system resilience is a complex multifacet problem. It requires integrated approaches combining different methods and tools for modelling, verification and quantitative analysis.

Methods and Techniques for Formal Development and Quantitative Assessment

In this chapter, the approaches and tools that relied on the modelling, verification and assessment of resilient distributed systems is described.

Development Methodologies

Development of a resilient system is a challenging engineering task that can be significantly facilitated by the use of formal model-based techniques. It allows the developers to build a system in a *rigorous* way and verify that the system specification meets the requirements. Moreover, formal modelling facilitates systematic derivation of fault tolerance mechanisms and complex reconfiguration solutions.

Traditionally, the methods that have rigorous mathematical basis are called *formal methods*. Formal techniques provide the developers with a strong mathematically-grounded argument about correctness of the system design. The main idea behind the formal modelling and verification is to rely on mathematics and formal logic to avoid imprecision, ambiguity, incompleteness or misunderstanding of system requirements described in natural language [19], and enable formal verification guaranteeing the system under consideration system model adheres to the given specification. Unlike testing, formal techniques allow us to ensure full coverage of possible system behaviours for achieving system resilience.

Traditionally, we distinguish between *proof-based* and *model checking* approaches. The general idea behind the automated proof-based verification is following: for given a mathematical or logical statement a computer program (prover) attempts to construct a proof that the statement is true. Typically, *theorem proving* approaches are used to ensure that a model satisfies the desired system properties. Verification is performed without actual model execution or simulation; therefore it allows us to explore the full model state space with respect to the specified properties. Some well-known examples of theorem proving software systems are Isabelle [20, 21], Coq [22], PVS [23, 24], Z3 [25], CVC3 [26, 27, 28], Vampire [29, 30], etc.

In contrast to the proof-based approach, *model checking* is a verification technique that explores all possible system states in a brute-force manner [31, 32]. Specifically, a model checker examines system scenarios in a systematic manner and, thereby, shows whether a given system model satisfies a certain property. Model checking helps us to find violation of the property in specifications by providing counterexamples. There is a big variety of model checkers tools that can be used in verification, e.g., SPIN [33, 34], UPPAAL [35], ProB [36], PRISM [37].

Formal methods are successfully applied in development and verification of complex dependable systems [38]. They are used in such domains as transportation systems [39, 40, 41], space and avionic system [42, 43, 44], traffic management and signaling systems [45, 46], medical devices [47, 48], etc.

Significant advances in integrating formal methods to industrial practice have been achieved in the Deploy project [49]. The project has advanced development of the industrial-strength platform Rodin for state-based modelling and verification of complex resilient systems in the Event-B formalism [50]. This has motivated the choice of Event-B as the formal development framework to be employed.

Event-B Method

In this section, formal development framework - Event-B is presented. The Event-B formalism - a variation of the B Method [51] - is a state-based formal approach that promotes the correct-by-construction development approach and verification by theorem proving [50]. The Event-B framework was influenced by the Action Systems [52, 53, 54] - a formal approach to model distributed, parallel, and reactive systems.

Modelling in Event-B. In Event-B, a system model is specified using the notion of an *abstract state machine* [50]. An abstract state machine encapsulates the model state represented as a collection of model variables, and defines operations on the state. Therefore, *machine* describes the *behavior* of the modelled system. A machine may also have the accompanying

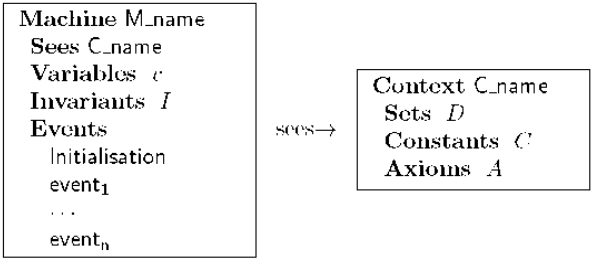


Figure 13.1: Event-B machine and context

component, called *context.* A context may include user-defined carrier sets, constants and their properties formulated as model *axioms.* A general form of the Event-B models is given in Figure 13.1.

An Event-B machine has a name Mname. The model state variables, v, are declared in the Variables clause and initialized in the

Initialization event. In Event-B, the model variables are strongly typed by the constraining predicates I called *invariants* given in the Invariants clause. The invariants also specify the properties that should be preserved during the system execution. The dynamic system behavior is defined by the set of atomic *events* specified in the Events clause. An event is essentially a *guarded command* that, in the most general form, can be defined as follows:

$$event \ \widehat{=} \ any \ \text{vl} \ where \ G \ then \ R \ end$$

where *vl* is a list of new local variables, G is the event *guard*, and R is the event *action.*

The guard is a state predicate that defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution non- deterministically. If none of the events is enabled then the system deadlocks. The occurrence of events represents the observable behavior of the system.

In general, the action of an event is a parallel composition of deterministic or non-deterministic assignments. A deterministic assignment, x := E(x,y), has the standard syntax and meaning. A non-deterministic assignment is denoted either as x :e S, where S is a set of values, or x :| P(x, y, x'), where P is a predicate relating initial values of x,y to some final value of x'. As a result of such a non-deterministic assignment, x can get any value belonging to S or according to P.

The semantics of Event-B actions is defined using so called *before-after* (BA) predicates [50]. A before-after predicate describes a relationship between the system states before and after execution of an event, as shown in

Table 13.1. Here x and y are disjoint lists (partitions) of the state variables, and x', y' are their values in the after-state.

Table 13.1: Before-after predicates

Action (R)	*BA(R)*
$x := E(x,y)$	$x' = \mathrm{E}(x,y) \cup y' = y$
$x :\in S$	$\exists z \bullet (\mathrm{z} \in S \cap x' = \mathrm{z}) \cap y' = y$
x :| P(x, y, x′)	$\exists z \bullet (P(x,z,y) \cap x' = z) \cap y' = y$

Event-B Refinement. Event-B employs a top-down *refinement-based* approach to the system development. Development in Event-B starts from an abstract system specification that models the most essential functional requirements. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions. Refinement usually affects both the context and the machine. Context refinement is a simple extension of the current context achieved by adding new constants, sets and axioms. A machine can be refined in two possible ways either using *data refinement* or *superposition refinement*. In particular, we can replace abstract variables by their concrete counterparts, i.e., perform *data refinement.* In this case, the invariant of the refined machine formally defines the relationship between the abstract and concrete variables. Via such a *gluing* invariant - "refinement relation" - we mathematically establish a correspondence between the state spaces of the refined and the abstract machines.

During *superposition refinement,* new implementation details are introduced into the system specification by means of *new* events and *new* variables. These new events can not affect the variables of the abstract specification and only define computations on newly introduced variables.

The new events correspond to the stuttering steps that are not visible at the abstract level, i.e., they refine implicit *skip.* To guarantee that the refined specification preserves the global behavior of the abstract machine, we should demonstrate that the newly introduced events *converge*. To prove it, we have to define a *variant* - an expression over a finite subset of natural numbers - and show that the execution of new events decreases it. Sometimes, convergence of an event cannot be proved due to a high level of non-determinism. In that case, the event obtains the status *anticipated*. This obliges the designer to prove, at some later refinement step, that the event indeed converges.

The correctness and consistency of Event-B models, i.e., verification of the model well-formedness, invariant preservation, deadlock-freeness, correctness of the refinement steps, is demonstrated by proving the relevant verification theorems - *proof obligations.* Proof

obligations are expressed as logical sequences, ensuring that the transformation is performed in a correctness-preserving way [50].

Modelling, refinement and verification of Event-B models is supported by an automated tool - Rodin platform [55]. The platform provides the designers with an integrated modelling environment, supports automatic generation and proving of the necessary proof obligations by means of wide range of automated provers. Moreover, various plug-ins created for Rodin platform allow a modeler to transform models from one representation to another, e.g. from UML to Event-B language [56, 57], or from Event-B specification to programming languages C/C++ [58, 59], ADA [60, 61], etc.

The Event-B refinement process allows us to gradually introduce implementation details, while proving preservation of functional correctness. Such an approach seamlessly weaves verification into the model development and allows us to construct detailed models of complex systems is highly automated incremental manner. By providing an immediate feedback on the correctness of model transformations, it helps us to cope with complexity of the system development. Another important mechanism for handling complexity of formal development is *decomposition*. Model decomposition helps the designers to separate component development from the overall system model but ensure that the components can be recomposed into overall system in a correctness-preserving way [62]. Event-B is equipped with three forms of decomposition: shared-variable [63, 64, 65], shared-event [65] and modularization [66], all of which are supported by the corresponding Rodin plug-ins [67, 68]. In this section we rely on a modularization extension of Event B [66].

Modularization. Modularization extension allows the designers to decompose a system into *modules*. Modules are components containing groups of callable atomic operations [66, 68]. Modules may have their own (external and internal) state and invariant properties. In general, they can be developed separately and then composed with the main system, when needed. Since decomposition is a special kind of refinement, such a model transformation is also a correctness-preserving step that has to be proven by discharging the relevant proof obligations.

A module description consists of two parts - *module interface* and *module body*. Let *M* be a module. A module interface is a separate Event-B component that has the unique name *MIjname.* A module interface consists of the external module variables *w,* the module invariants MI_Inv(c, s, w), and a collection of module operations, characterized by their pre- and postconditions defined in the Operations clause. In addition, a module interface may contain a group of standard Event-B events under the Processes clause.

Interface MI_name
Sees IC name
Variables w
Invariants MI_Inv(c, s, w)
Initialisation …
Processes
P_1 = any vl where g(c, s, vl, w) then S(c, s, vl, w, w') end
…
Operations
O_1 = any p pre Pre(c, s, vl, w) post Post(c, s, vl, w, w') end
…

Figure 13.2: Module interface

These events model autonomous module thread of control, expressed in terms of their effect on the external module variables. In other words, they describe how the module external variables may change between operation calls. The overall structure of a module interface is shown on Fig.13.2.

A formal development of a module starts with the deciding on its interface. Once an interface is defined, it cannot be changed in any manner during the development. This ensures that a module body may be constructed independently from a system model that relies on the module interface. A *module body* is an Event-B machine. It implements the interface by providing a concrete behavior for each of the interface operations. To guarantee that each interface operation has a suitable implementation, a set of additional proof obligations are generated. When the module M is imported into another Event-B machine

(specified by a special clause Uses), the importing machine may invoke the operations of M and read the external variables of M.

We can create several instances of the given module and import them into the same machine. Different instances of a module operate on disjoint state spaces. Identifier prefixes can be supplied in the Uses clause of the importing machine to distinguish the variables and the operations of different module instances or those of the importing machine and the imported module. Alternatively, the pre-defined constant set can be supplied as a additional parameter. In the latter case, module instances are created for each element of the given set, thus producing an indexed collection (array) of module instances. A detailed description of indexed modules is given in [69].

The modularization extension of Event-B facilitates formal development of complex systems by allowing the designers to decompose large specifications into separate components and verify system-level properties at the architectural level. As a result, proof-based verification as well as reliance on abstraction and decomposition adopted in Event-B offers the designers a scalable support for the development of complex distributed systems.

Quantitative Assessment

Formal modelling in Event-B allows the designers to derive complex system architecture, formulate and prove logical system properties and formally verify correctness of the system behavior. While functional correctness constitutes an important aspect of resilience, we also need to provide the developers with techniques for quantitative resilience assessment. Quantitative assessment plays an important role in the process of resilient system development because it allows the developers to predict the impact of changes on such vital aspects as, e.g., reliability and performance. Moreover, quantitative analysis helps to find suitable trade-offs between these properties as well as evaluate the impact of different architectural alternatives on system resilience. Therefore, we investigate possibility of integration of formal development in Event-B with quantitative resilience assessment. In particular, we study integration with the probabilistic symbolic

model checker PRISM [70], and discrete event simulation in SimPy[71].

PRISM model checker

PRISM model checker [70] is one of the leading software tool for formal modelling and verification of systems that exhibit probabilistic behavior. It provides support for analysis of three types of Markov process - discrete time Markov chains (DTMC), continuous-time Markov chains (CTMC) and Markov decision processes (MDP). Additionally, it supports modelling of (priced) probabilistic timed automata and st7]ochastic games (as a generalization of MDP) [72]. The state-based modelling language of PRISM relies on the *reactive modules* formalism [.

A PRISM model consists of a number of *modules* which can interact with each other. The behavior of each module is described by a set of guarded commands that are quite similar to Event-B events. The latter fact significantly simplifies transformation of Event-B machines to the corresponding PRISM specifications.

While analyzing a PRISM model, one can define a number of temporal logic properties to be evaluated by the tool. To assess resilience, we can rely on verifying the time-bounded reachability and reward properties. In the property specification language of PRISM, they can be formulated using the supported temporal logics - PCTL (Probabilistic Computation Tree Logic) [73] for discrete-time models and CSL (Continuous Stochastic Logic) [74, 75] for continuous-time models.

Similarly to Event-B, the PRISM language is a high-level state-based modelling language. Essentially, PRISM supports the use of constants and variables. The variables in PRISM are finite-ranged and strongly typed. They also can be global or local, i.e., associated with a particular *module.*

A PRISM specification is constructed as a parallel composition of modules that can be synchronized using the standard CSP parallel composition. In addition to local variables, each module has a number of guarded commands that determine its dynamic behavior. Each command consists of a guard and one or more updates over local and

global system variables. Each update is annotated with a probabilistic weight (in discrete-time models) or rate (in continuous-time models). Similarly to events in Event-B, a guarded command can be executed (i.e., is enabled) only if its guards evaluate to TRUE. If several guarded commands are enabled at the same time, then the choice between them is defined by the model type - it is non-deterministic for MDP models, probabilistic for DTMC models or modelled as an (exponential) race condition for CTMC models.

PRISM tool has been successfully employed in many domains including distributed coordination algorithms, wireless communication protocols, security as well as dependability and biological models.

To enable probabilistic analysis of Event-B models in PRISM, we rely on the continuous-time probabilistic extension of the Event-B framework [76, 77]. This extension allows us to annotate actions of all model events with real-valued rates and then transform a probabilistically augmented Event-B specification into a continuous-time Markov chain. It also implicitly introduces the notion of time into Event-B models: for any state, the sum of action rates of all enabled in these state events defines a parameter of the exponentially distributed time delay that takes place before some enabled action is triggered.

Discrete-event simulation

Due to similarity between PRISM and Event-B languages, the translation from a Event-B model to a PRISM specification is rather straightforward. It makes the use of PRISM model checker attractive for the performing quantitative assessment. However, the model checking technique does not always scale to large applications. In such case, *simulation* offers a viable alternative for quantitative analysis of resilience.

Traditionally, *simulation* is called the process of imitating how an actual system behaves over time [78]. A simulation generates an artificial system history, thereby enabling analysis of system general behavior. Simulation is built around the notion of *event* - an occurrence that changes the state of the system. The system state variables are viewed as a collection of all information that is required to define what is happening within the system at a given moment of time.

A widely-used type of simulation is known as discrete-event simulation (DES). In a DES, system state remains constant over an interval of time between two consecutive *events*. Thus events signify occurrences that change the system state. Events can be classified as either internal or external. *Internal events* occur within the modelled system, while *external events* occur outside the system, but still might affect it. A simulation is run by a mechanism that repeatedly moves simulated time forward to the starting time of the next scheduled event, until there are no more events [79].

From the architectural perspective, a DES system consists of a number of *entities* (e.g., components, processes, agents, etc.), which are either producers or recipients of discrete events. Static entities (e.g., queues, buffers, etc.) can often be represented as *resources.* Resources have limited availability, leading to competition among entities. Waiting for a particular event to occur can lead to a *delay*, lasting for an indefinite amount of time. In other cases, the time estimate may be known in advance. Events can be also *interrupted* and pre-empted, e.g., in reaction to component failures or pre-defined high-priority events.

There are four primary simulation paradigms [78]: process-interaction, event-scheduling, activity scanning, and the three-phase method. We use SimPy [71] - a simulation framework based on process- interaction in Python. Essentially, SimPy is a discrete-event simulation library written in Python. The behaviour of active entities (e.g., customers, requests) is modelled by means of *processes*. All processes settle in an environment and interact with the environment and with each other via events.

Processes are described by simple Python generators. During their lifetime, they create events and yield them in order to wait for them to be triggered. When a process yields an event, the process gets suspended. SimPy resumes the process, when the event occurs. *Timeout* is an important event type. Events of this type are triggered after a certain amount of (simulated) time has passed. SimPy also provides various types of shared resources to model limited capacity congestion points (like servers, queues, buffers, etc.).

Discrete-event simulation represents an attractive technique for quantitative evaluation of different system characteristics. It allows the

designers to perform various “what-if” type of analysis that demonstrates sensitivity of the service architecture to changes of its parameters. For instance, it gives an understanding on how the system reacts on peak-loads, how adding new resources affects its performance, what are the relationships between the degree of redundancy and fault tolerance, etc. Moreover, while simulating the behavior, the designers can also obtain the information on which parameters should be monitored at run-time to optimize a resource allocation strategy. However, to obtain all the above-mentioned benefits, the designers have to ensure that the simulation models are correct and indeed representative of the actual system. In particular, this can be achievable via integration of simulation technique with formal modelling.

Advancement questions

1. What is the concept of the System Resilience?
2. Explain the concept of Dependability and name its key attributes.
3. What are the four main techniques on which engineering dependable systems are able to rely?
4. Why Goal-Based Development is crucial for software engineering? p. 133
5. What does agent concept provide us?
6. What is the concept of the System Autonomy and Reconfiguration and what were the reasons (motives) to remove humans from the control?
7. What is the main idea behind the formal modelling and what are its main approaches?
8. Explain the process of modelling in Event-B method
9. What are the two possible ways of Event-B Refinement ?

10. What is the role of Quantitative Assessment and what are the tools to integrate Event-B method with quantitative resilience assessment?

REFERENCE

1. Pereverzeva Inna Formal Development of Resilient Distributed Systems / Inna Pereverzeva // PhD diss., Turku Centre for Computer Science, Abo Akademi University, Faculty of Science and Engineering, Joukahaisenkatu, Turku, Finland. – 2015.
2. Laprie J.-C. Resilience for the Scalability of Dependability : In Fourth IEEE International Symposium on Network Computing and Applications / J.-C. Laprie // IEEE .-2005
3. Laprie J.-C. From Dependability to Resilience / J.-C. Laprie // IEEE Computer Society .- 2008.
4. Strigini L. Resilience: What is it, and how much do we want? / L. Strigini // IEEE Security & Privacy .- 2012 .-10(3) .-72-75
5. Avizienis A. Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr // IEEE Trans. Dependable Sec. Comput.-2004.-1(1).-11–33
6. Avizienis A. Dependability and its Threats - A taxonomy: In IFIP Congress Topical Sessions / A. Avizienis, J.-C. Laprie, and B. Randell .-2004.- pages 91– 120
7. Guelfi N. An Introduction to Software Engineering and Fault Tolerance, chapter Software Engineering of Fault Tolerant Systems / N. Guelfi, P. Pelliccione, H. Muccini, and A. Romanovsky // Series on Software Engineering and Knowledge Eng.- 2007

8. Lamsweerde Axel van. Goal-oriented requirements engineering: A guided tour / Axel van Lamsweerde // In Requirements Engineering.- 2001.- pages 249–263
9. Lamsweerde Axel van. Requirements Engineering: From System Goals to UML Models to Software Specifications / Axel van Lamsweerde // Wiley.-2009.
10. Lamsweerde A. van From system goals to software architecture. In Formal Methods for Software Architectures, Third International School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures / A. van Lamsweerde .- Bertinoro, Italy : Springer.- September 22-27 .-2003 .- pages 25–43.
11. Franklin S. Is it an agent, or just a program?: A taxonomy for autonomous agents: In Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, ECAI ’96 / S. Franklin and A. Graesser // Springer-Verlag.- 1997.- pages 21–35.
12. Ferber J. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence / J. Ferber.- Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.- 1999.-1st edition .
13. Jennings N. R. On agent-based software engineering / N. R. Jennings // Artif. Intell.-2000.- 117(2).-277–296.
14. OMG Mobile Agents Facility (MASIF): http://www.omg.org.
15. Kephart J. O. The vision of autonomic computing / J. O. Kephart and D. M. Chess // IEEE Computer.-2003. -36(1).-41–50.
16. Fisher M. Verifying autonomous systems / M. Fisher, L. A. Dennis, and M. P. Webster // Commun. ACM .- 2013.-56(9).-84–93

17. Dennis L. A. Reconfigurable autonomy / L. A. Dennis, M. Fisher, J. M. Aitken, S. M. Veres, Y. Gao, A. Shaukat, and G. Burroughes // KI .- 2014.- 28(3).-199–207
18. Huebscher M. C. A survey of autonomic computing - degrees, models, and applications / M. C. Huebscher and J. A. McCann // ACM Comput. Surv.- 2008.-40(3),
19. RushbyJ. Formal methods and the certification of critical systems: Technical Report SRI-CSL-93-7 / J. Rushby // Computer Science Laboratory, SRI International.- 1993
20. The HOL System : online at http://www.cl.cam.ac.uk/research/hvg/HOL/.
21. NipkowT. Isabelle/HOL — A Proof Assistant for Higher-Order Logic: volume 2283 of LNCS / T. Nipkow, L. C. Paulson, and M. Wenzel // Springer .- 2002
22. Coq. The Coq Proof Assistant: online at https://coq.inria.fr/what-iscoq
23. Owre S. PVS: A prototype verification system: In Automated Deduction - CADE-11, 11th International Conference on Automated Deduction /S. Owre, J. M. Rushby, and N. Shankar // Saratoga Springs, Springer.- NY, USA.- June 15-18.- 1992, pages 748–752
24. Owre S. PVS: combining specification, proof checking, and model checking / S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas .-New Brunswick, NJ, USA : In Computer Aided Verification, 8th International Conference, CAV '96, Springer.- July 31 - August 3.-1996 pages 411–414
25. Mendon¸ca de Moura L. Z3: an efficient SMT solver. In Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008 / L. Mendon¸ca de Moura and N.

Bjørner.- Budapest, Hungary: Springer.- March 29-April 6.- 2008 .-pages 337– 340

26. Barrett C. CVC3. In Werner Damm and Holger Hermanns, editors, Proceedings of the 19th International Conference on Computer Aided Verification (CAV ’07): volume 4590 of Lecture Notes in Computer Science / C. Barrett and C. Tinelli.- Berlin, Germany: Springer-Verlag.- 2007.- pages 298–302
27. Stump A. A decision procedure for an extensional theory of arrays:In Proceedings of the 16th IEEE Symposium on Logic in Computer Science (LICS ’01) / A. Stump, D. L. Dill, C. W. Barrett, and J. Levitt .- Boston, Massachusetts :IEEE Computer Society.- 2001 .- pages 29–37
28. Berezin S. A practical approach to partial functions in CVC Lite. In Selected Papers from the Workshops on Disproving and the Second International Workshop on Pragmatics of Decision Procedures (PDPAR ’04), volume 125(3) of Electronic Notes in Theoretical Computer Science / S. Berezin, C. Barrett, I. Shikanian, M. Chechik, A. Gurfinkel, and D. L. Dill // Elsevier .- 2005.-pages 13–23
29. Riazanov A. Vampire. In Automated Deduction - CADE-16, 16th International Conference on Automated Deduction : Proceedings, volume 1632 of Lecture Notes in Computer Science / A. Riazanov and A. Voronkov.- Trento, Italy: Springer.- 1999 .-pages 292–296.
30. Riazanov A. The design and implementation of VAMPIRE / A. Riazanov and A. Voronkov // AI Commun.- 2002.-15(2-3).-91–110
31. Baier C. Principles of Model Checking / C. Baier and J.-P. Katoen // MIT press.- 2008

32. Clarke E. M. The birth of model checking. In 25 Years of Model Checking - History, Achievements, Perspectives / E. M. Clarke // SpringerVerlag Berlin Heidelberg.- 2008.-pages 1–26.
33. Holzmann G. J. The model checker SPIN. /G. J. Holzmann// IEEE Trans. Software Eng.- 1997.-23(5).-79–295
34. Holzmann G. J. The SPIN Model Checker - primer and reference manual. /G. J. Holzmann. //Addison-Wesley.-2004.
35. Bernardo M. Formal Methods for the Design of Real-Time Systems/M. Bernardo, F. Corradini, and K. G. Larsen, editors//volume 3185 of Lecture Notes in Computer Science: Springer .-2004.
36. The ProB Animator and Model Checker:online at http://www.stups.uni-duesseldorf.de/ProB/index.php5/.
37. PRISM. Probabilistic Symbolic Model Checker. online at http://www.prismmodelchecker.org/.
38. Woodcock J. C. Formal methods: Practice and Experience/J. C. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald.// ACM Comput. Surv.- 2009.-41(4)
39. Badeau F. Using B as a high level programming language in an industrial project: Roissy VAL. /F. Badeau and A. Amelot.//In Formal Specification and Development, 4th International Conference.-Guildford:Springer .-2005.- pages 334–354
40. Industrial Use of the B Method.: http://www.methode-b.com/wpcontent/uploads/2012/08/ClearSy-Industrial Use of B1.pdf.
41. Behm P. A successful application of B in a large project. /P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. M´et´eor //In FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems.- France: Springer.-1999.-pages 369–387.

42. S. P Miller. Specifying the mode logic of a flight guidance system in core and scr. /S. P Miller. // In Proceedings of the 2nd Workshop on Formal Methods in Software Practice, volume 4916 of Lecture Notes in Computer Science:ACM.- 1998.-page 44–53.
43. Esteve M.-A. Formal correctness, safety, dependability, and performance analysis of a satellite. /M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein.//In Proceedings of the 34th International Conference on Software Engineering:Piscataway.-2012.- pages 1022–1031,
44. Rushby J. M. Analyzing cockpit interfaces using formal methods. /J. M. Rushby.//Electr. Notes Theor. Comput. Sci..-2001.-43.-1–14
45. H¨orl J. Formal specification of a voice communication system used in air traffic control./J. H¨orl and B. K. Aichernig. //In FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems: Springer.- 1999.-page 1868
46. Bacherini S. A story about formal methods adoption by a railway signaling manufacturer. S. Bacherini, A. Fantechi, M.Tempestini, and N.Zingoni. In Formal Methods, 14th International Symposium on Formal Methods, volume 4085 of Lecture Notes in Computer Science:Springer .-2006.-pages 179–189
47. Bowen J. Safety-critical systems, formal methods and standards. Software Engineering Journal/J. Bowen and V. Stavridou.-1993. -8(4).-189–209.
48. Praful Jetley R. A case study on applying formal methods to medical devices: computer-aided resuscitation algorithm /R. Praful Jetley, C. Carlos, and S. Purushothaman Iyer //STTT.-2004.-5(4).-320–330.

49. Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). IST FP7 IP Project: online at http://www.deploy-project.eu/.
50. J.-R. Abrial Modeling in Event-B. / J.-R. Abrial.-Cambridge University Press.- 2010
51. Abrial J.-R. The B-Book: Assigning Programs to Meanings. /J.-R. Abrial.-Cambridge University Press.-2005
52. Back R. J. R. Decentralization of Process Nets with Centralized Control. / J. R. Back and R. Kurki-Suonio //In ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing.-1983.- pages 131–142
53. Back R. J. R. Stepwise Refinement of Action Systems. Structured Programming/R. J. R. Back and K. Sere.-1991.-12(1).-17–30
54. Back R. J. R. From Action Systems to Modular Systems. Software – Concepts and Tools/R. J. R. Back and K. Sere.-1996.-17(1).-26–39
55. Rodin. Event-B Platform:online at http://www.event-b.org/.
56. Snook C. F. UML-B: formal modeling and design aided by UML. ACM Trans./C. F. Snook and M. J. Butler.// Softw. Eng. Methodol.- 2006.-15(1).-92–122
57. M. Y. Said, M. J. Butler, and C. F. Snook. Language and tool support for class and state machine refinement in UML-B. /M. Y. Said, M. J. Butler, and C. F. Snook.//volume 5850 of Lecture Notes in Computer Science:Springer.- 2009.-pages 579–595
58. M´ery D. Automatic code generation from event-b models./ D. M´ery and N. K. Singh. // In Proceedings of the Second Symposium on Information and Communication Technology:ACM.- 2011.-pages 179–188

59. Mery D. Transforming event b models into verified c# implementations. /D. Mery and R. Monahan. //In Alexei Lisitsa and Andrei Nemytykh, editors, VPT 2013, volume 16 of EPiC Series .-2013.-pages 57–73
60. Edmunds A. Formal modelling for ada implementations: Tasking event-b./A. Edmunds, A. Rezazadeh, and M. J. Butler.// In Reliable Software Technologies International Conference on Reliable Software Technologies: Springer.-2012.-pages 119–132
61. Edmunds A. Templates for event-b code generation. / A. Edmunds.//In Abstract State Machines, Lecture Notes in Computer Science.- France:Springer.- 2014.-pages 284–289
62. Hoang T. S. A survey on event-b decomposition./ T. S. Hoang, A. Iliasov, R.Silva, and W. Wei.//ECEASST.- 2011.-46
63. Abrial J.-R. Refinement, decomposition, and instantiation of discrete models: Application to event-b. Fundam./J.-R. Abrial and S. Hallerstede.// Inform.- 2007.-77(1-2).-1–28
64. Hoang T. S. Event-b decomposition for parallel programs. /T. S. Hoang and J.-R. Abrial.//In Abstract State Machines,Second International Conference.- Canada : Proceedings.-2010. -pages 319–333
65. Butler. M. J. Decomposition structures for event-b./M. J. Butler// In Integrated Formal Methods, 7th International Conference.-Germany:Du¨sseldorf.- 2009. -pages 20–38
66. Iliasov A. Supporting Reuse in Event B Development: Modularisation Approach./A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala.// In Proceedings of Abstract State Machines:Springer.-2010.- pages 174–188.

67. Silva R. Decomposition tool for event-b. /R. Silva, C. Pascal, T. S. Hoang, and M. J. Butler //Softw., Pract. Exper.- 2011.-41(2).-199–208.
68. RODIN Modularisation Plug-in: Documentation at http://wiki.eventb.org/index.php/Modularisation Plug-in.
69. Iliasov A. Support of Indexed Modules in Event-B. /A. Iliasov, L. Laibinis, E. Troubitsyna, and A. Romanovsky.//In Proceedings of the 4th Rodin User and Developer Workshop:TUCS Lecture Notes.- 2013 .-pages 29–30.
70. Kwiatkowska M. PRISM 4.0: Verification of Probabilistic Real-time Systems./ M. Kwiatkowska, G. Norman, and D. Parker// In CAV'11, International Conference on Computer Aided Verification:Springer.- 2011.-pages 585–591.
71. SimPy. Simulation framework in Python:online at http://simpy.readthedocs.org/.
72. Kwiatkowska M. Advances in Probabilistic Model Checking./ M. Kwiatkowska and D. Parker// In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, Software Safety and Security - Tools for Analysis and Verification, volume 33 of NATO Science for Peace and Security Series - D: Information and Communication Security:IOS Press.- 2012.-pages 126–151
73. Hansson H. A Logic for Reasoning about Time and Reliability / H. Hansson and B. Jonsson// In Formal Aspects of Computing.-1994.-pages 512–535
74. Aziz A. Verifying Continuous Time Markov Chains./A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. // In CAV'96, International Conference on Computer Aided Verification:Springer.-1996.-pages 269–276
75. Baier C. Approximate Symbolic Model Checking of Continuous-Time Markov Chains/C. Baier, J.-P. Katoen, and

H. Hermanns. // In CONCUR'99, International Conference on Concurrency Theory:Springer.-.-pages 146–161

76. Tarasyuk A. Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B/ A. Tarasyuk, E. Troubitsyna, and L. Laibinis.// In IFM 2012, Integrated Formal Methods:Springer.-2012.-pages 237–252
77. Tarasyuk A. Integrating stochastic reasoning into event-b development. /A. Tarasyuk, E. Troubitsyna, and L. Laibinis. //Formal Asp. Comput.- 2015.-27(1).-53– 77
78. Schriber T. J. How discrete-event simulation software works. / T. J. Schriber and D. T. Brunner.//In Jerry Banks, editor, Handbook of Simulation,John Wiley & Sons .-2007.- pages 765–812.

CHAPTER 14. FORMAL DEVELOPMENT AND QUANTITATIVE ASSESSMENT OF RESILIENT DISTRIBUTED SYSTEMS

Content of the chapter 14

14.1 Overview of the Proposed Approach

This chapter presents an integrated approach to development and assessment of resilient distributed systems. Approach relies on formal development by refinement in Event-B, which is augmented with quantitative resilience assessment in the probabilistic model checker PRISM and discrete event simulation in SimPy.

Unreliability of system components and communication channels, complex component interactions as well as highly dynamic operating conditions make the problem of developing resilient distributed systems challenging. To address this problem, we need advanced methods that are able to cope with the complexity inherent to such systems.

The Event-B framework relies on three main mechanisms for coping with complexity: abstraction, decomposition and proofs. Development of a distributed system in Event-B starts from creating an abstract system specification (model). Often such a specification gives a “black-box” model of the system behavior, i.e., it focuses on defining the externally observable behavior while abstracting away from the system component architecture and the internal functional behavior. The initial Event-B model represents a centralized system that exhibits the desired externally observable behavior and properties. The following refinement steps aim at transforming the abstract model into a detailed system specification by gradually unfolding the system architecture, precisely defining the functional behavior as well as deriving a detailed representation of component interactions.

In this chapter, we show how the described above generic approach to development of distributed systems by refinement can be tailored to support resilience-explicit development of different types of systems. The resulting approach shares the common idea of using refinement as the main vehicle for unfolding the system architecture and dynamics. Refinement facilitates systematic introduction of the mechanisms for ensuring system resilience while defining various inter-relationships between the system elements. Moreover, since quantitative assessment of different resilience characteristics is an essential part of the system design for resilience, we show how Event-B models can be augmented

with quantitative data and, as a result, serve as a basis for quantitative resilience assessment.

Resilience-explicit development based on functional decomposition. To present approach, let us start first by considering the systems that perform a predefined scenario that can be implemented by a deterministic sequential execution flow. Such kind of the system behavior is typical for a certain class of systems, which includes, among others, service-oriented and control systems. In service-oriented systems, a service can be often modelled as a sequential composition of subservices. Such a composite service can be provided only if each subservice is successfully executed. A similar type of reasoning can be used for modelling control systems. Since control systems are cyclic, each execution cycle can be represented as a sequential execution of certain functional blocks.

In approach, let us explicitly define the resilience-explicit refinement process for such systems. Specifically, we demonstrate that modelling of not only the nominal system behavior but also a possibility of system failures already at the abstract level can facilitate a rigorous systematic derivation of the required fault tolerance mechanisms. Then we discuss generic functional decomposition as a refinement step that results in defining a high-level execution flow. We explain how to establish a connection between a global system failure and the corresponding failures in the execution flow. Further, we demonstrate how refinement can be used for deriving the component-based system architecture and linking component failures with those in the system execution flow. Moreover, establishing the connection between the functionality of the system and that of its components allows us to systematically derive the system reconfiguration mechanisms that are based on reallocation of execution of certain functional tasks from the failed components to the healthy ones. Finally, to evaluate the impact of reconfiguration on the system performance and reliability, we augment the resulting Event-B models defining various reconfiguration scenarios with the necessary probabilistic information and demonstrate how to quantitatively assess different reconfiguration strategies.

Modelling component interactions. After presenting the generic resilience-explicit development process for systems with a deterministic sequential execution flow, we focus on detailed analysis of component interactions while providing a certain function (service) or participating in a specific collaboration. To perform the required functions while ensuring fault tolerance, the system components should interact and cooperate with each other. To facilitate reasoning about such cooperative behavior, we treat components as agents and a resilient distributed system as a multi-agent system. The multi-agent modelling perspective helps us to define the essential properties of cooperative agent activities. As a result, we derive the constraints that should be imposed on agent interactions to ensure correct and safe functioning despite component and communication failures.

Resilient-explicit goal-oriented refinement process. Another large class of distributed systems includes the systems whose execution flow is highly non-deterministic, with a loose connection between functional blocks. Typical examples of such systems are standard multi-agent systems whose components (agents) have some degree of autonomy. For such kind of systems, it is convenient to adopt the goal-oriented reasoning style. In the this section a development method for such systems that formalizes the resilient-explicit goal-oriented refinement process is proposed.

Resilience can be defined as the ability of a system to achieve its objectives - goals - despite failures and other changes. We define a set of specification and refinement patterns that reflect the main concepts of the goal-oriented development. The refinement approach is employed to support the goal decomposition process, thus allowing us to define the system goals at different levels of abstraction. Let us follow the same generic strategy for development of distributed goal-oriented systems by refinement. Namely, we start by abstractly defining system goals, then perform goal decomposition by refinement, and finally introduce a representation of system agents, whose collaborative activities ensure goal reachability. Therefore, resilience-explicit goal-oriented refinement approach aims at ensuring goal reachability “by construction”. It allows the developers to systematically introduce the required reconfiguration mechanisms to ensure that the system

progresses towards achieving its goals despite agent failures (thereby, address “negative” changes) or becomes more performant by using its agents more efficiently (thereby, address “positive” changes).

We consider a dynamic reconfiguration as a powerful technique for achieving system resilience because it allows the system to adapt to changes by modifying its structure, inter-agent relationships and dependencies. However, ensuring correctness of the incorporated reconfiguration mechanisms is a complex task. To address this issue, we formalize the possible interdependencies between goals and agents as well as formulate the conditions for ensuring goal reachability in a reconfigurable multi-agent system. The proposed formalization gives a formal systematization of the introduced concepts and can be seen as generic guidelines for designing reconfigurable systems.

In the resilience-explicit goal-oriented development approach let us assume that the agents are sufficiently reliable, i.e., some agents will stay operational during the whole process of goal achieving. To validate such an assumption and derive the constraints on agents reliability, we need to employ quantitative analysis.

Quantitative assessment is also required to evaluate the impact of various architectural solutions on the system performance and reliability. Integration with probabilistic model checking in PRISM allows us to achieve these objectives. We augment Event-B models with quantitative data and transform them into input models for the PRISM model checker. As a result, quantitative assessment allows the designers to make informed design choices and develop systems with predictable resilience characteristics.

Modelling and assessment of resilient architectures Finally, in the last part of this chapter, we investigate how a resilient-explicit refinement approach can be adopted to derive distributed architectures with the incorporated fault tolerance mechanisms. Let us consider a particular approach to ensure fault tolerance - write-ahead logging (WAL) - and experiment with deriving several alternative architectures implementing it. Each architectural solution exhibit different reliability and performance characteristics. Let us demonstrate how to derive different architectures by refinement and formally define data integrity

and consistency properties that logically formulate reliability characteristics.

Moreover, we propose a graphical notation which facilitates resilience assessment of architectural alternatives by discrete event simulation in SimPy - a library and development framework in Python. The quantitative analysis in SimPy allows us to evaluate the impact of a particular architectural solution on the system reliability/performance ratio.

14.2 Resilience-Explicit Development Based on Functional Decomposition.

In this section, the resilience-explicit refinement process for the systems that perform a certain predefined scenario is presented. The aim is to facilitate rigorous modelling of both nominal and off-nominal system behavior and to support structured derivation of a functional system specification that integrates the required fault tolerance mechanisms. This is achieved by an explicit representation of the failure behavior at all levels of abstraction.

Let us assume that the system under construction should provide a service that can be represented as a composition of certain functional blocks as shown in Fig.14.1.

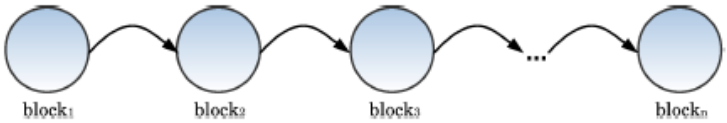


Figure 14.1: Generic execution control ow

In the context of service-oriented systems, the functional blocks correspond to subservices, while in the context of control systems they represent the steps of a single iteration of a control loop. The resulting sequence of functional blocks defines the system execution flow.

In the initial specification, presented in Fig.14.1, let us abstractly model the changing status of service execution. Initially the system is idle and can be activated to provide a service, as modelled by the event Activation. This results in changing the value of the boolean variable

idle from *TRUE* to *FALSE.* Upon service activation, the event Execution becomes enabled. It models the progress of service execution by non-deterministically changing the value of the variable *process.* The set PSTATES = {*FINISHED, UNFINISHED, ABORT}* represents the possible status of the service execution process. The value of *process* remains *UNFINISHED* until all the functional blocks of the service are successfully executed. Upon completion of the service execution, the variable *process* obtains value *FINISHED*, which in turn enables the event Finish. This event changes the status of the system to inactive, i.e., the variable idle obtains the value *TRUE.*

The value *ABORT* of the variable process designates the occurrence of an unrecoverable failure. The corresponding event Abort deadlocks the specification, i.e., models the fact that the software halts its execution.

Machine abs behavior	Execution ≙
Variables idle; process	where idle = FALSE ^ process =
Invariants	UNFINISHED
idle ∈ BOOL	then process : ∈ PSTATES
process ∈ PSTATES	end
idle = TRUE => process =	Finish ≙
UNFINISHED	where idle = FALSE ^ process =
Events	FINISHED
Activation ≙	then idle; process = TRUE;
where idle = TRUE	UNFINISHED
then idle := FALSE	end
end	Abort ≙
	when process = ABORT
	then skip
	end

Figure 14.2: Abstract System Behavior Model

Functional Decomposition by Refinement. Next we refine the abstract specification by introducing an explicit representation of the execution flow, i.e., by modelling an execution sequence of the predefined functional blocks. For illustrative purposes, we consider only a simple case of sequential execution. However, in general, we can

define any complex scenario (including branching, rollbacking, etc.) by formulating the corresponding axioms in the model context.

For simplicity, we assume that the "id" of each block is defined by its execution order, i.e. $block_1$ is executed first and so on. To explicitly model the impact of failures of individual block executions on the overall service provisioning, we define the following function *block_state*:

$$\text{Block\_state} \in 1..n \rightarrow \text{BSTATES},$$

where BSTATES = *[NI, OK, POK, NOK}* is an enumerated set of the possible status values for block execution. Initially, none of the block is executed, i.e., the status of each block is *NI*. The block execution can lead to successful completion *(block-state* gets the value *OK*), an unrecoverable failure *(NOK*), or a failure that can be recovered by resigning the block to a different available component *(POK*).

To model functional decomposition, we replace the abstract variable *process* by the variable *block_state*, i.e., perform data refinement. The gluing invariants for such data refinement are given below. An excerpt from the refined specification is shown in Fig. 14.3.

The introduced events Start and Progress model execution of the corresponding functional blocks. They specify the process of sequential selection of one block after another until all blocks are executed, i.e., the service is completed, or execution of some blocks fails, i.e., service provisioning fails. The sequential order between the events is enforced by the corresponding guards. In particular, the guards ensure that the execution of all previous blocks has been successful completed.

Machine ref1 Variables idle; block state Invariants … Events // First block execution Start ≙ refines Execution any res when res ∈ {OK; POK;NOK} ∧ block state(1) 6≠ OK ∧ block state(1) 6≠ NOK	// block execution Progress ≙ refines Execution any j; res where j > 0 ∧ j < n ∧ res ∈ {OK; POK;NOK} ∧ block state(j) = OK ∧ block state(j + 1) 6≠ OK ∧ block state(j + 1) 6≠ NOK then block state(j + 1) := res end …

then block state(1) := res end	

Figure 14.3: Flow Modelling

We formulate and prove the following invariants defining some essential properties of the defined execution flow. The properties postulate that a next block can be chosen for execution only if execution of all the previously chosen blocks was successfully completed and, moreover, the subsequent block was not executed yet:

$$\forall l \bullet \mathrm{I}\ G\ 2\,..\,\mathrm{n} \wedge \text{block etate(l)} \neq \mathrm{NI} => (\forall i \bullet i \in 1\,..\,l - 1 = > \text{block\_state(i)} = \mathrm{OK}),$$

$$\forall l \bullet l\ G\ 1\,..\,n - 1 \wedge \text{block state(l)} \neq OK => (\forall i \bullet i \in l + 1\,..\,n => \text{block\_state(i)} = NI).$$

The refined model should guarantee that the execution process progresses towards completion of service provisioning. This is ensured by the gluing invariants that establish the relationship between the abstract specification and the functional decomposition introduced by refinement:

$$block_{state[1\,..n]} = \{OK\} \Rightarrow \text{process} = \text{FINISHED},$$

$$(block\_state[1\,..\,n] = \{\mathrm{OK}, \mathrm{POK}, \mathrm{NI}\} \vee \text{block\_state}[1\,..\,n] = \{\mathrm{NI}\}) \Rightarrow$$

$$process = UNFINISHED, (\exists \mathrm{i} \bullet \mathrm{i} \in 1\,..\,\mathrm{n} \wedge block\_\text{state(i)} = NOK \Rightarrow process = ABORT.$$

Component Modelling. The purpose of the *Component Modelling* refinement step is to link the functional blocks with the corresponding system components that are responsible for executing them.

We define a variable representing the current state for each system component:

$$\text{comp\_state} \in \text{COMPONENTS} \rightarrow \text{CSTATES},$$

where COMPONENTS represents the set of all system components, while CSTATES stands for an enumerated set *{NA, OPERATIONAL, FAILED}*. A component has a status *NA* if it is not currently involved into the execution process. A healthy active component has the status *OPERATIONAL*, while a failed component obtains the status *FAILED*.

To define the relationship between the functional blocks and the components that are responsible for executing them, we introduce the variable *exec*:

$$\text{exec} \in \text{COMPONENTS} \leftrightarrow 1..n.$$

Here we do not impose any additional restrictions on the "component- block" interdependency. However, one can specify a certain condition that should hold during the execution, e.g., postulate that a component should be responsible for executing at least one block. We refine the events modelling three cases of the block execution. Below, Fig. 14.4 presents the events modelling the successful and unrecoverable failed execution of blocks.

// successful block execution SuccessProgress ≙ refines Progress any j, c when j > 0 ^ j < n ^ block state(j) = OK ^ block state(j + 1) 6≠ OK ^ block state(j + 1) 6≠ NOK comp state(c) = OPERATIONAL then block state(j + 1) := OK exec := exec ∪ {$c \mapsto j + 1$} end	// unrecoverable failure FailProgress ≙ refines Progress any j, c when j > 0 ^ j < n ^ block state(j) = OK ^ block state(j + 1) 6≠ OK ^ block state(j + 1) 6≠ NOK comp state(c) = FAILED then block state(j + 1) := NOK exec := exec ∪ {$c \mapsto j + 1$} end

Figure 14.4: Component Modelling

To link the status of block execution with the status of the component responsible for executing it, we formulate and prove the invariant, establishing relationship between them:

$$\forall i \bullet i \in 1..n \wedge \text{block\_state(i)} = \text{OK} \Longrightarrow (\exists c \bullet c \in \mathit{COMPONENTS} \wedge (c \mapsto i) \in \text{exec} \wedge \text{comp\_state(c)} = \text{OPERATIONAL}).$$

Abstract Reconfiguration Modelling. At this refinement step we introduce an abstract model of reconfiguration that is performed by reassigning responsibility to execute a certain functional block from a failed component to a healthy one. In particular, we define a new function variable *assign* to represent a block assigned to be executed to a component:

$$assign \in COMPONENTS \leftrightarrow 1..n.$$

We add new events AssignFirstBlock and AssignBlock modelling block assignment (see Fig. 4.5). In the guard of the events SuccessStart, FailStart, SuccessProgress and FailProgress, we add the additional conditions where we check that the corresponding block has been assigned before to the component. In our modelling, we assume that a component may fail only during its block execution.

```
// block assignment
AssignBlock ≙
any j, c
when …
comp_state(c) =
OPERATIONAL
j ∉ ran(assign) ∧
c ∉ dom(exec) ∧ c ∉
 dom(assign)
then assign := assign ∪{c ↦
j}
    end
```

```
// successful block execution
    SuccessProgress      ≙ refines
SuccessProgress
    any j, c
    when …
    (c ↦ j+1) ∉ assign ∧
    comp_state(c)                =
OPERATIONAL
    then block_state(j + 1) := OK
    exec := exec ∪{c ↦ j}
    assign := assign \ {c ↦ j+1}
    end
```

Figure 14.5: Block Reallocation Modelling

Let us note that in our specification we model error detection in a highly abstract way. However, we can further refine the model to elaborate on the involved error detection mechanism, for instance, by defining component pinging.

The proposed resilience-explicit refinement process is generic. It abstracts away from the concrete functionality that the system under construction should implement and defines only what kind of the

refinement steps should be performed and which types of properties that should be defined and verified. The final refinement step can be seen as a starting point for introducing different reconfiguration strategies and, consequently, employing quantitative assessment technique.

Each reconfiguration alternative (i.e., a different reconfiguration strategy or mechanism) results in creating the corresponding Event-B model. To evaluate the impact of different reconfiguration alternatives, we transform the models into inputs to the PRISM model checker. To achieve this, we augment the corresponding Event-B models with the following probabilistic data:

- the lengths of time delays required by components to execute specific functional blocks;
- the occurrence rates of possible failures of these components.

Moreover, we replace all the local nondeterminism with the (exponential) race conditions. Such a transformation allows us to represent the behavior of Event-B machines by continuous time Markov chains and use the probabilistic symbolic model checker PRISM to evaluate reliability and performance of the proposed models.

14.3 Modelling Component Interactions with Multi-Agent Framework

In the resilience-explicit refinement process presented above, we abstracted away from modelling component interactions while performing the predefined functions. Usually, execution of a certain functional block and especially achieving fault tolerance relies on the assumption that the components behave in a cooperative way. For instance, when execution of a functional block is being reallocated from a failed component to a healthy one, the healthy component needs to accept the new responsibility, i.e. behave cooperatively.

The multi-agent modelling paradigm facilitates reasoning about the cooperative component behavior. We adopt this paradigm to demonstrate how to reason about resilience of complex component interactions. It allows us to treat the components of a resilient distributed system as agents and execution of system functions or

services as cooperative agent activities. Next let us present the formal approach to resilience-explicit modelling of agent interactions.

Let us formally reason about agents, their attributes and behavior as well as agent cooperative activities. The formalization allow us to establish logical connections between agents and define the conditions under which agents interactions result in correct execution of a cooperative activity. Moreover, the established dynamic connections (called *relationships*) between agents allow us to explicitly reason about resilience of complex agent interactions.

A multi-agent system *MAS* is a tuple $(A, \mu, R, \Sigma, E, Active, Rel)$, where A is a set of all the system agents, μ is the system middleware, R is a set of all possible relationships between agents in a MAS, Σ is the system state space, and E is a collection of system events (reactions). Moreover, the dynamic system attributes Active and Rel map a given system state to a set of the active (healthy) system agents and a set of dynamic relationships between the active agents respectively.

The system dynamics is modelled as a set of system events E, where each event e e E can be formally represented as a relation on input and output system states, i.e., $e : \Sigma \leftrightarrow \Sigma$. The dynamic system attributes *Active* and *Rel* are then simply functions from Σ, i.e., *Active* : $\Sigma \rightarrow$ P(A) and *Rel* : $\Sigma \rightarrow$ P(R), returning respectively the current sets of active system agents and dynamic relationships between them. Intuitively, two or more system agents being in a dynamic relationship means that these agents are currently involved in a specific collaboration needed to provide a predefined system function or service.

Each system agent belongs to a particular agent class. Essentially these classes represent a partitioning of the system agents into different groups according to their capabilities. In general, there can be many agent classes A_i, $i \in 1..n$, such that A_i C A. We assume that all of them are disjoint.

The system middleware μ can be considered as a special kind of the system agent that is always present in the system. The main responsibility of the middleware is to ensure communication between different agents, detect appearance of new agents or disappearance (both normal

and abnormal) of the existing agents, recover from the situations when the required connections between the agents are lost, etc.

The system state space Σ consists of all possible states of agents and the middleware. The system events E then include all internal and external system reactions (state transitions). We assume that each agent may have a number of dynamic attributes that can be changed during these transitions. The values of these attributes in a particular state also determine whether a particular agent is currently "eligible", i.e., can be involved in execution of specific system events.

Each *interaction activity* between different agents (or an agent and the middleware) may be composed of a set of events. Moreover, system events may model appearance or disappearance of agents, sending request from one agent to another, recovery of lost connections, etc.

A set R defines all possible dynamic relationships or connections between agents of the same or different classes. We assume the existence of a number of available data constructor functions to create elements of R. More precisely, for each relationship $r \in \mathrm{R}$, r is modelled as a result of an application of some data constructor function

$$\mathrm{r} = \mathrm{R}_{\mathrm{Constr_i}}(a_1, \mathrm{a}_2, \dots, \mathrm{a}_\mathrm{m}),$$

where $R\_Constr_i : A^*_{i1} x A^*_{i2} \dots x A^*_{im} \mapsto R$ for some $m \in \mathrm{N_l}$ and each $A^*_{ij} = \mathrm{A_k} \cup \{?\}$ for some agent type A_k. Here $\mapsto$ designates an injection function and "?" stands for an unknown agent of the corresponding class.

A relationship can be *pending*, i.e., incomplete. This is indicated by putting the question marks instead of a concrete agent, e.g., $R\_Constr_i(\mathrm{a}_1, \mathrm{a}_2, ?, \mathrm{a}_4, ?)$. Pending relationships are often caused by disappearance or a failure of the agents previously involved in a relationship. Moreover, an existing active agent may initiate a new pending relationship. Once a pending relationships is resolved (completed), the question mark is replaced by a concrete agent.

While R represents all possible agent relationships, *Rel* stores the currently active (both complete and pending) relationships. For a relationship to be active, all the involved in it agents should be active as well. In other words, for any $\sigma \in \Sigma$ and $\mathrm{r} \in \mathrm{Rel}(\sigma)$, if a concrete agent a_i is involved in r, it should be an active one, i.e., $a_i \in Active(\sigma)$.

Let us now consider some expected properties that should be hold for interactions between agents as well as between agents and the middleware.

Property 14.1.

Let EAA and $EA\mu$, be all interaction activities (sets of events) defined between agents or between agents and middleware respectively. Moreover, for each agent $a \in$ A, let *Ea* be a set of events in which the agent *a* might be involved. Then

$$\forall \sigma, a * \ a \ \in \ Active\ (\sigma) \ \Rightarrow \ \mathrm{Ea}(\sigma) \ \in \ EAA \ \mathrm{U} \ EA\mu,$$

$$and$$

$$\forall \sigma, \mathrm{a} * \ a \ \notin \ \mathrm{Active}\ (\sigma) \ \Rightarrow \ Ea(\sigma) \ \in \ \mathrm{EA\mu}$$

The property restricts agent interactions with respect to the agent activity status. For instance, this property implies that, when an agent is recovering from a failure, it cannot be involved into any cooperative activities with other agents. Therefore, while modelling agent interactions, we have to take into account the agent status.

To represent such a behavior in Event-B, we define the following events modelling agent activities with the middleware. In particular, the events Appearance and Disappearance model joining and leaving the system by agents (of any classes).

Appearance ≙ any a when a ∈ AGENTS ∧ $a \notin Active$ then Active := Active ∪{a} end	Disappearance ≙ any a when a ∈ Active then Active := Active \ {a} end

Here *AGENTS* defines a set of all system agents (i.e., A), while *Active* represents the subset of active agents.

In a similar way, only active agents can interact with each other as shown by the event Interaction.

Interaction ≙ any a_1,a_2 *when* $a_1 \ \in \ Active \ \wedge \ a_2 \ \in \ Active \ \wedge$

$Elig_1(a_1) = TRUE \wedge Elig_2(a_2) = TRUE \wedge ...$
then ...
end

Here $Elig_1(a_1)$ and $Elig_2(a_2)$ abstractly model specific eligibility conditions on the agents that should be checked before their interaction.

The next expected property concerns collaborative activities between the agents and how these activities are linked with the inter-agent relationships.

Property 14.2.

Let EAA be all the interactions in which active agents may be involved. Moreover, for each active agent a, let R_a be all the relationships it may be involved in. Finally, for each collaborative activity $CA \in$ EAA, let A_{CA} be a set of all agents involved in it. Then, for each $CA \in EAA$ and $a_1, a_2 \in A_{CA}$,

$$R_{a1} \cap R_{a2} \neq \emptyset$$

This property restricts the interactions between the agents: only the agents that are linked by dynamic relationships (some of which may be pending) can be involved into cooperative activities.

To specify abstractly a collaborative activity between agents in Event-B, we define an event CollabActivity. In the event guard, we check that both agents, participating in collaboration, are active, eligible to be involved, and there is a pre-existing relationships that permits their interactions:

CollabActivity $\widehat{=}$
any a_1, a_2
when
$a_1 \in$ Active $\wedge$ $a_2 \in$ Active $\wedge$
$Elig_1(a_1) = TRUE \wedge Elig_2(a_2) = TRUE \wedge$
$RConst_i(a_1 \mapsto a_2) \in Rel$
then …
end

Here $RConst_i$ is a data constructor for a specific kind of agent relationships, which is formally specified in the model context. In a similar way, we can model collaborating activities involving any number of agents.

We can specify initiation of a new relationship between agents in two ways. In the case, when all the required agents are active, eligible and ready to enter the relationship, it can be defined by the following event InitiateRelationship.

InitiateRelationship ≙
any a_1, a_2
when $a_1 \in$ Active $\wedge$ $a_2 \in$ Active $\wedge$
Ellig(a1) = TRUE $\wedge$ Ellig(a2) = TRUE
then Rel := Rel $\cup$ $RConst_i(a_1 \mapsto a_2)$
end

The opposite situation, when some agent of the initiated relationship is still unknown, can be defined by the following event InitiatePendingRelationship. Here the pre-defined element *None*, *None* e *AGENTS*, is used to designate a missing agent in the pending relationship (i.e., the special agent "?" in the above formalization). In the event shown below, an agent a_1 initiates a new pending relationship, where the place for a second agent of the particular type is currently vacant (i.e., is marked by *None*). The resulting pending relationships are added to *Rel*.

InitiatePendingRelationship ≙
any a_1
when $a_1 \in$ Active $\wedge$ Ellig(a_1) = TRUE
then Rel := Rel $\cup$ $RConst_i(a_1 \mapsto$None)
end

Essentially, all the relationships containing *None* in the place of any their elements denote pending relationships.

To resolve the pending relationship *RConsti(a1 ∧ None)*, the corresponding agent has to join this collaborative activity. This situation is abstractly modelled by the event AcceptRelationship.

AcceptRelationship ≙
status anticipated
any a_1; a_2

when $a_1 \in Active \wedge a_2 \in Active \wedge Ellig(a_2) = TRUE \wedge RConsti(a_1 \mapsto None) \in Rel$
then $Rel := (Rel \setminus RConst_i(a_1 \mapsto None)) \cup RConst\_i(a_1 \mapsto a_2)$
end

The system middleware p keeps track of the pending relationships and tries to resolve them by enquiring suitable agents to confirm their willingness to enter into a particular relationship. We can also distinguish a special subset of the pending relationships that have a priority over the others. These relationships are linked with executing critical functions, and hence called critical. A responsibility of the middleware is to detect situations when some of the established or to be established relationships become pending and guarantee eventual resolution of them. Essentially, this means that no pending request is ignored forever and the middleware tries to enforce the given preferences, if possible.

While developing a resilient MAS, we should ensure that all high priority relationships will be established. Therefore, we have to verify that corresponding cooperative activities, establishing these critical relationships, once initiated, are successfully completed. More precisely, we have to verify the following property:

Property 14.3.

Let EAA_{crit}, where $EAA_{crit} \subseteq EAA$, be a subset containing critical collaborative activities. Moreover, let R_{pen} and R_{res}, where $R_{pen} \subseteq R$ and $R_{res} \subseteq R$, be the subsets of pending and resolved relationships defined for these activities. Finally, let R_{CA}, where $CA \in EAA$ and $R_{CA} \subseteq R$, be all the relationships the activity CA can affect. Then, for each activity $CA \in EAA_{crit}$ and relationship $R \in R_{CA}$,

$$(R \in R_{pen}) \rightsquigarrow (R \in R_{res}),$$

where $\rightsquigarrow$ denotes "leads to" operator.

This property postulates that eventually all the pending relationships should be resolved for each cooperative activity.

To verify this property in Event-B, we have to prove that the event AcceptRelationship converges, i.e., eventually gets enabled. We achieve this by requiring that, at the abstract level, the event AcceptRelationship has the *anticipated* status. This means that

"resolving" of a pending relationship is postulated rather than proved. However, at some refinement step, this event status also obliges us to prove that the event or its refinements *converge*, i.e., to prove that the process of resolving a relationship will eventually terminate.

.

14.4 Goal-Oriented Modelling of Resilient Systems

In this section, we propose the resilience-explicit refinement process that aims at facilitating development of complex distributed systems whose execution flow is highly non-deterministic, with a loose connection between functional blocks. Typical examples of such systems are multi-agent systems. For such kind of systems, it is convenient to adopt the goal-oriented reasoning style. Goals provide us with a suitable basis for reasoning about system resilience. Indeed, resilience can be considered as an ability of a system to achieve its objectives - goals - despite failures and changes.

14.5 Pattern-Based Formal Development of Resilient MAS

To support the goal-oriented development of multi-agent resilient systems in Event-B, we define a set of Event-B *specification* and *refinement patterns* that reflect the main concepts of the goal-oriented engineering. Patterns define generic reusable solutions that facilitate development of complex systems [1, 2, 3, 4].

In the context of formal development in Event-B, patterns represent generic modelling solutions that can be reused in similar developments via instantiation. Usually, an Event-B pattern contains abstract types, constants and variables. The context component of such a model defines the properties that should be satisfied by concrete instantiations of abstract data structures. Moreover, the invariant properties of a pattern, once proven, remain valid for all instantiations.

Let us assume that we have defined a collection of Event-B patterns: P_1, P_2,..., P_n that refine each other in the following way: P_1 is refined by P_2, …, P_{n-1} is refined by P_n.

Such a refinement chain expresses a generic development by refinement. Abstract data structures of all the involved patterns become generic parameters of the development. Each pattern abstractly defines

a solution for specifying a certain modelling aspect. The initial pattern *P*l presents a generic model (specification pattern), serving as a starting point of such a development. Each refinement step focuses on formulating specific modelling aspects that should be introduced as a result of the corresponding refinement transformation. The result of such a refinement transformation is called a *refinement pattern*.

The proposed *specification and refinement patterns* interpret some essential activities of the goal-oriented engineering within the Event-B refinement process:

- *Goal Modelling Pattern*: explicitly defines high-level system goal(s) in Event-B and postulates goal reachability;
- *Goal Decomposition Pattern*: demonstrates how to define the system goals at different levels of abstraction in Event-B (i.e., how to decompose high-level system goal(s) into subgoals). An application of the pattern results in introducing a goal hierarchy;
- *Agent Modelling Pattern*: allows the designers to introduce agents into a specification and associate them with the system goals;
- *Agent Refinement Pattern*: explicitly defines the static and dynamic agent characteristics (attributes).

Goal Modelling Pattern. We use the concept of a state transition system to reason about the system behavior. To formulate the *Goal Modelling Pattern,* we start by introducing an abstract type *GSTATE* defining the system state space. Moreover, *Goal* is a non-empty subset of *GSTATE* that abstractly defines the given system goal(s). We say that the system has achieved the desired goals if its current state belongs to *Goal*.

While modelling a system in Event-B, we should ensure that the system under development achieves the desired goal. We can formally express this by requiring that the system terminates in a state belonging to *Goal.* The process of accomplishing such a goal is modelled by the event Reaching_Goal. The event is enabled while the goal is not reached. The variable *gstate* might eventually change its value from not reached to reached (i.e., *gstate* becomes G *Goal),* thus designating achievement of the goal:

Reaching_Goal ≙

status anticipated *when*
gstate ∈ GSTATE \ Goal *then*
gstate: ∈GSTATE
end end

The system terminates when Reaching_Goal event becomes disabled, i.e., when a state satisfying *Goal* is reached. Note that the event Reaching_Goal has the status *anticipated*. Hence, at this stage reachability is postulated rather than proved, postponing the proof of convergence to some later refinement step. However, later when we introduce more system details, we will be able to prove that the event (or one of its refinements) *converges*.

Goal Decomposition Pattern. The main idea of goal-oriented development is to decompose the high-level system goals into a set of corresponding subgoals. Essentially, the resulting subgoals define intermediate stages of the process of achieving the main goal(s). The objective of the *Goal Decomposition Pattern* is to explicitly introduce such subgoals into the system specification.

While defining the lower-level goals, we should ensure that the high- level goals remain achievable. Hence our refinement pattern should reflect the relation between the high-level goals and their subgoals. Moreover, it should ensure that high-level goal reachability is ensured and can be defined via reachability of the corresponding lower-level subgoals. We assume that the subgoals are independent of each other. This means that reachability of any subgoal does not affect reachability of another one.

To model this pattern in Event-B, we assume (for simplicity, and without losing generality) that the system goal *Goal* is achieved by reaching three subgoals. The subgoals are defined as corresponding variables: *Subgoal1, Subgoal2*, and *Subgoal3.* The goal independence assumption allows us to partition the high-level goal state space *GST AT E* into three non-empty subsets: *SG-STATE1, SG-STATE2* and *SGSTATE3.*

The following mapping function *State.map* establishes the gluing relationship between the new state spaces *SG-STATEi, i* ∈ 1..3, and the abstract state space:

Statemap G SG-STATE1 x SG-STATE2 x SG-STATE3 $\twoheadrightarrow$GSTATE.

Here $\twoheadrightarrow$ designates a bijection function. Essentially it partitions the original goal state space into three independent parts.

To postulate interdependence between reachability of the main goal and that of its subgoals, we rigorous express the following property: the main goal is reached if and only if all three subgoals are reached:

$$\forall sg1, sg2, sg3 \cdot sg1 \in \text{Subgoal1} \wedge sg2 \in \text{Subgoal2} \wedge sg3 \in \text{Subgoal3} \Leftrightarrow \text{State\_map}(sg1 \mapsto sg2 \mapsto sg3) \in \text{Goal}.$$

In general, we can logically relate the main goal with any expression on its subgoals.

A refinement step performed according to the *Goal Decomposition Pattern* is an example of Event-B data refinement. We replace the abstract variable *gstate* with the new variables $gstate_i \in SGSTATEi$, $i \in 1..3$. The new variables model the state of the corresponding subgoals. The following gluing invariant allows us to prove data refinement:

$$\text{gstate} = \text{State\_map}(\text{gstate1} \mapsto \text{gstate2} \mapsto \text{gstate3}).$$

Now the event Reaching_Goal of the abstract machine is decomposed into three similar events Reaching_SubGoalij $i \in 1..3$, modelling the process of achieving of the corresponding subgoals, as shown below:

Machine M_GD Reaching_SubGoal1 ≙ refines Reaching_Goal status *anticipated* when gstate1 ∈ SG_STATE1 \ Subgoal1 *then* gstate1 : ∈ SG_STATE1 *end*

The proposed *Goal Decomposition Pattern* can be repeatedly used to refine subgoals into the subgoals of a finer granularity until the desired level of detail is reached.

Agent Modelling Pattern. The proposed *Abstract Goal Modelling* and *Goal Decomposition* patterns allow us to specify the system goal(s)

at different levels of abstraction. In multi-agent systems, (sub)goals are usually achieved by system components - agents, which are independent entities that are capable of performing certain tasks. In general, the system might have several types of agents that are distinguished by the type of tasks that they are capable of performing. Our next refinement pattern - *Agent Modelling Pattern* - allows us to model system agents and associate them with goals.

We introduce the set *AGENTS* that abstractly defines the set of system agents. Additionally, we distinguish three non-empty sets *EL-AG1*, *EL_AG2*, and *EL_AG3* of the agents that are capable of achieve the corresponding subgoals.

Agent might fail while trying to achieve a certain subgoal. To reflect this in the specification, we introduce dynamic sets of the eligible agents represented by the variables $elig_i$, $elig_i \subseteq EL\_AGi$, where $i \in 1..3$. We say that an agent is eligible to perform a certain goal if it is active and capable to accomplish it.

Agent failures have direct impact on the process of subgoals achievement, i.e., the goal assigned to the failed agent cannot be reached. To reflect this assumption in our model, we refine the abstract event Reaching_SubGoali by two events SuccessfuLReaching-SubGoali and Failed_Reaching_SubGoali, $i \in 1..3$, which respectively model the successful and unsuccessful reaching of the subgoal by some eligible agent, as shown below:

Machine M_AM

Successful_Reaching_SubGoal$_1$ $\widehat{=}$ refines Reaching_SubGoal$_1$

status convergent

any ag

when

$gstate_1 \in SG\_STATE_1 \setminus Subgoal_1 \wedge ag \in elig_1$

then

$gstate_1 :\in Subgoal1$

end

Failed Reaching SubGoal$_1$ $\widehat{=}$ refines Reaching SubGoal$_1$

status convergent

any ag

when
$gstate_1 \in SG\_STATE_1 \setminus Subgoal_1 \wedge ag \in elig_1 \wedge card(elig_1) > 1$
then
$gstate_1 :\in SG\_STATE_1 \setminus Subgoal_1$
$elig_1 := elig_1 \setminus \{ag\}$
end

In the guard of the event Failed_Reaching_SubGoal;L, we restrict possible agent failures by postulating that at least one agent associated with the subgoal remains operational: *card(elig1)* > 1. This assumption allows us to change the event status from *anticipated* to *convergent*. In other words, we are now able to prove that, for each subgoal, the process of reaching it eventually terminates. In practice, the constraint to have at least one operational agent associated with our model can be validated by probabilistic modelling of goal reachability, which we discuss later in this chapter.

Agent Refinement Pattern. In the *Agent Modelling Pattern,* we have defined the notion of agent eligibility quite abstractly by formulating the relationship between subgoals and the corresponding agents that are capable of achieving them. Our *Agent Refinement Pattern* aims at elaborating on the notion of agent eligibility. We introduce agent attributes - *agent types* and *agent statuses*, and redefine an eligible agent as an operational agent that belongs to a particular agent type.

We define an enumerated set of agent types *AG_TYPE={ TYPE1, TYPE2, TYPE3}* and establish the correspondence between abstract sets of agents and the corresponding agent types by the following axioms:

$$\forall ag \cdot ag \in EL\_AGi \Leftrightarrow atype(ag) = TYPEi, i \in 1..3.$$

We consider an agent as capable to perform a certain subgoal if it has the type associated with this subgoal.

To model explicitly the dynamic operational status of each agent, we add a new variable *astatus*:

$astatus \in AGENTS \rightarrow AG\_STATUS$.

Here set *AG_STATUS = {OK, KO},* where *OK* and *KO* designate operational and failed agents correspondingly.

Now we can data refine the abstract variables $elig_i$, $i \in 1..3$. The following gluing invariants associate them with the concrete sets:

$elig_i = \{a \mid a \in AGENTS \wedge atype(a) = TYPE_i \wedge astatus(a) = OK\}$, for $i \in 1..3$.

In our case, the dynamic set of agents eligible to perform a certain subgoal becomes a set of active agents of the particular type. The event Failed_Reaching_SubGoali is now refined to take into account the concrete definition of agent eligibility. The event also updates the status of the failed agent.

Successful_Reaching_SubGoal1 = refines Successful_Reaching_SubGoal1 any *ag* when
gstatel ∈ SG_STATE1 \ Subgoall ∧ astatus(ag) = OK ∧ atype(ag) = TYPE1 *then*
gstate1 *: G* Subgoal1 *end*
Failed_Reaching_SubGoal1 = refines Failed_Reaching_SubGoal1
any *ag* when
gstate1 ∈ SG_STATE1 \ Subgoal1 ∧ astatus(ag) = OK A atype(ag) = TYPE1 ∧
card({a\a ∈ AGENTS ∧ atype(a) = TYPE1 ∧ astatus(a) = *OK}) > 1 then*
gstate1 *:* ∈ SG_STATE1 \ Subgoali || astatus(ag) *:*= KO *end*

As mentioned above, to prove the defined goal reachability property, we had to make the assumptions related to agent reliability, i.e., assume that some agents remain operational to successfully complete the goal achieving process. To validate this assumption, we can employ quantitative assessment - probabilistic model checking techniques. To enable probabilistic analysis of Event-B models in the probabilistic model checker PRISM, we rely on the continuous-time probabilistic extension of the Event-B framework [5]. The idea of this approach is as follows. We annotate actions of all model events with real-valued rates (e.g., failure rate, service rate) and then transform such a probabilistically augmented Event-B specification into a continuous-time Markov chain, which we represent in PRISM. Then we can assess the probability of achieving the goal as well as to compare several alternative system configurations.

The resilience-explicit goal-oriented refinement approach presented above allowed us to identify the key concepts required for formal development of resilient MAS. It has inspired as to propose a conceptual framework for goal-oriented reasoning about resilient MAS that puts a specific emphasis of rigorous definition of system reconfigurability. Next we overview the proposed formalization.

14.6 Formal Goal-Oriented Reasoning About Resilient Reconfigurable MAS

In this section let us overview proposed formalization of the reconfigurability concept within a multi-agent goal-oriented framework. The aim is to gradually define the notions of system goals and agents together with their different interrelationships. Let us systematically introduce the necessary constraints on the system dynamics to facilitate derivation of a necessary reconfiguration mechanism. Here we consider reconfigurability as an ability of agents to redistribute their responsibilities and associations to ensure goal reachability.

Goal-oriented State Transition System. We start by extending the standard definition of a state transition system (including the set of all system states V, the next-state relation *Trans*, and the set of initial system states *Init*) with the notion of goals that a system is trying to accomplish. More specifically, we introduce the set of all possible system goals G and the function *GMap* mapping a given system goal to a subset of system states:

$$\mathrm{GMap} : \mathcal{g} \rightarrow \mathrm{P}(\Sigma).$$

Essentially, the function *GMap* assigns semantics to any goal from G by associating it with a non-empty set of states (a predicate) of Σ.

Further let us extend the goal-oriented state transition system with the notion of subgoals. To introduce inter-relationships between the system goals, e.g., distinguishing particular goals and their subgoals, we define two structures - the relation on goals *G_graph* and the function *SGMap*:

$\mathrm{G\_graph} : \mathcal{g} \leftrightarrow \mathcal{g}$ and $\mathrm{SGMap} : \mathcal{g} \rightarrow \mathrm{P}(\Sigma)$.

Essentially, *G_graph* describes relationships between different goals, e.g., how a particular goal can be decomposed into its subgoals

and so on. *SGMap(g)* stands for mapping an arbitrary expression on subgoals of g into a set of states, corresponding to achieving the parent goal g. Intuitively, *SGMap*(g) stands for the necessary precondition for achieving goal g.

Essentially, achieving any of subgoals must contribute to reaching the parent goal:

$$\forall g, g' : \mathcal{g}. g' \in Subgoals(g) \Rightarrow SGMap(g) \cap GMap(g') \neq \emptyset,$$
$$\text{where } Subgoals(g) = \{g' : \mathcal{g} \mid (g \mapsto g') \in G\_graph\}.$$

Introducing Agents. Next, we extend a goal-oriented state transition system by introducing agents that can carry out tasks required for achieving the system goals. We introduce the type (set) *A* for all possible system agents and define the function *Active* to distinguish a subset of active agents in the current system state:

$$Active : \Sigma \rightarrow \mathrm{P}(\mathrm{A}).$$

By "active" agents we mean the agents that can carry out the tasks in order to achieve the system goals. In its turn, the inactive agents are either the agents which are not currently present in the system or those which failed and thus incapable to carry out any tasks.

To reflect the heterogeneous nature of multi-agent systems, next we introduce possible agent attributes. Namely, we associate certain classes of agents with specific types of system goals they are able to accomplish. To formalize it, we first introduce classifications of system agents and goals and then define relationships between the introduced classes.

The following functions

$$atype : \mathrm{A} \rightarrow AType \text{ and } gtype : \mathcal{g} \rightarrow GType,$$

associate each agent and goal with their respective type, where *AType* and *GType* are abstract types containing all possible agent and goal types respectively.

The separate goals of the same goal type can be achieved independently, i.e., can be assigned to different agents that work in parallel to accomplish them:

$$\forall \mathrm{g1}, \mathrm{g2} : \mathcal{g}, gt : GType. gtype(g1) = gt \wedge gtype(g2) = gt \wedge \mathrm{g1} \neq \mathrm{g2} \wedge GMap(g1) \cap GMap(g2) = \emptyset,$$

To represent interrelationships between different agent and goal types, we introduce the relation *AG_Rel:*

$$AG\_Rel: AType \leftrightarrow GType.$$

This formalizes a connection between the corresponding agent and goal types clarifies which agents can be given the tasks related to specific system goals.

Agent Subordination and Supervision. Defining agent types and hierarchy of goal types allows us to introduce a subordination structure between agent types. Essentially, subordination means that one agent may be the "master" (manager) of the other agent(s). Naturally, agent subordination supposes that some agents "supervise" activities of other agents. Moreover, a supervising agent can give concrete goal assignments to subordinate agents, which, in turn, should "report" to its supervisors once the assigned goal has been accomplished. The unreached system goals can be also dynamically partitioned among the supervisor agents, essentially modelling accepted responsibilities of those agents for supervision over some goals.

To introduce such subordination, we define a relation on agent types, called *ASub*:

$$A\_Sub : AType \leftrightarrow AType.$$

Moreover, for each pair of subordinated agent types, there should exist (at least one) pair of the related goal types such that goals of the parent goal type can be handled by agents of the "master" agent type, while goals of the subgoal type can be handled by agents of the subordinate agent type.

Let us note that the introduced notions of agent types, subordination, ability to accomplish or supervise a particular goal, constitute *static* properties of a multi-agent goal-oriented system. On the other hand, since agents can change their active/inactive status during system execution, the function *Active* expresses a dynamic system characteristic. To formally define a system configuration and the corresponding reconfiguration mechanism for tolerating system changes, we need to define additional *dynamic* system characteristics. First, in a specific dynamic system state, a particular agent can be

attached to another agent, which serves as its supervisor. A specific goal that has not been yet reached can be put under *responsibility* of a particular supervisor agent. Moreover, a specific goal can be *assigned* by a supervisor to one of its subordinate agents. Later, the assigned goal can be executed by the corresponding agent. If the agent fails to achieve the assigned task, its goal can be reassigned to another agent capable to achieve it.

We formulate these dynamic notions formally. For instance, *agent attachment* is defined as the function *Attached*, such that

- $Attached : \Sigma \rightarrow P(A \; x \; A)$,
- $\forall \sigma : \Sigma, a1, a2 : A.(a1 \mapsto a2) \in \text{Attached}(\sigma) \wedge$
- $a1 \in \text{Active}(\sigma) \wedge a2 \in \text{Active}(\sigma) \wedge \text{atype}(a1) \mapsto \text{atype}(a2) \in A_{\text{Sub}} \wedge \neg(\exists a3 : A.a3 \neq a1 \wedge (a3 \mapsto a2) \in \text{Attached}(\sigma))$.

Therefore, for any agents a_1, a_2 and system state σ, the expression ($a_1 \mapsto a_2$) e *Attached(σ)* implies that (i) both agents are active in σ, (ii) the agent type of a_2 is subordinate to that of a_1, and (iii) the agent a_2 is not currently attached to any other supervisor agent.

Moreover, a goal-oriented multi-agents system supports agent attachment if, at any point where the conditions for agent attachment are satisfied, the system has an opportunity (but not an obligation) to do such an action.

Similarly to *agent attachment*, we define *goal responsibility* (the corresponding function called *Responsible*) and *goal assignment* (the corresponding function called *Assigned*) as system dynamic attributes (i.e., they depend on the current system state). Goal responsibility specifies the relationships between certain goals and the agents currently supervising them. In its turn, *goal assignment* defines the relationships between the goals and pair of agents that supervise and perform these goals respectively. Moreover, a goal-oriented multi-agents system supports goal responsibility and goal assignment if, at any point where the conditions for these properties are satisfied, the system is able to do these actions.

Now, the introduced above notions and characteristics allow us to define notion of a reconfigurable system and reason about system reconfigurability.

Reasoning about System Reconfiguration Towards Goal Achievement. Based on the definitions, we can explicitly define multi-agent systems that support system dynamic reconfiguration. Specifically, these are the systems that allow redistributing (unassigned) goals to different responsible agents or reattaching (unassigned) agents to different supervisor agents. Moreover, the following properties must hold:

$$\forall \sigma : \Sigma, \mathrm{g} : \mathcal{g}, : a1, a2 : A.(\mathrm{g} \mapsto a1) \in \mathrm{Responsible}(\sigma) \wedge \mathrm{gtype}(\mathrm{g}) \in \mathrm{AS\_goals}(\mathrm{atype}(\mathrm{a2})) \wedge \neg(\exists a3 : A.(\mathrm{g} \mapsto a1 \mapsto a3)) \in \mathrm{Assigned}(\sigma)) \Rightarrow \exists \sigma' : \Sigma.(\sigma \mapsto \sigma') \in \mathrm{Trans} \wedge (\mathrm{g} \mapsto a2) \in \mathrm{Responsible}(\sigma')$$

and

$$\forall \sigma : \Sigma, al, a2, a3 : A.(\mathrm{al} \mapsto a2) \in Attached(\sigma) \wedge (atype(a3)\ atype(a2)) \in A\_Sub \wedge \neg(\exists g : \mathcal{g}.(\mathrm{g} \mapsto \mathrm{al} \mapsto \mathrm{a2})) \in Assigned(\sigma)) \Rightarrow \exists \sigma' : \Sigma.(\sigma \mapsto \sigma') \in \mathrm{Trans} \wedge (\mathrm{a3} \mapsto a2) \in \mathrm{Attached}(\sigma).$$

Essentially, these two properties require the existence of state transitions allowing to redistribute goal responsibility and agent attachment. Here the condition *gtype(g)* ∈ *AS_goals(atype(*a_2*))* checks that the type of the agent a_2 allows the agent to supervise the goal *g*, while the condition *(atype(*a_3*)* ↦ *atype(*a_2*))* ∈ *A_Sub* requires that the agent type of a_2 is subordinate to that of a_3.

Theorem: Goal reachability in a reconfigurable agent system.

For a *reconfigurable* goal-oriented multi-agent system (G, Σ, *Init, Trans, GMap, A, Active)*, the following property is true:

$$\forall \sigma : \Sigma, g : \mathcal{g}.\sigma \in \mathrm{dom}(Trans) \wedge \sigma \notin GMap(g) \Rightarrow \exists \sigma' : \Sigma.(\sigma \mapsto \sigma') \in Trans + \wedge\ \sigma' \in GMap(g).$$

Essentially, theorem states, that for a reconfigurable goal-oriented multiagent system, any goal that is not yet reached at any (non-final) system state is reachable. Let us note that the theorem is proved to formally demonstrate that all the introduced notions and mechanisms are sufficient to ensure goal reachability in such a system.

The goal-oriented framework provides us with a suitable basis for reasoning about reconfigurability. It allows us to define reconfiguration

as ability of agents to redistribute their responsibilities to ensure goal reachability. The proposed formal systematization of the involved concepts can be seen as generic guidelines for formal development of reconfigurable systems.

14.7 Modelling and Assessment of Resilient Architectures

In this section, we focus on the problem of formal modelling and quantitative assessment of resilient architectures. In particular, we experiment with different architectural alternatives implementing a well-known fault tolerant mechanism for distributed systems (WAL). Each alternative provide the developers with different reliability guarantees expressed in the terms of data consistency and data integrity properties. However, since higher reliability usually results in lower performance, it is desirable to quantitatively assess this ratio under different configurations parameters and loads.

Let us start this section by briefly describing the WAL mechanism. Then we demonstrate how to use the employed refinement approach to derive resilient architectures. Finally, let us propose a graphical notation facilitating construction and validation of models for resilience assessment in SimPy - a library and development framework in Python.

WAL mechanism and data base replication. The WAL mechanism is a standard data base technique for ensuring data integrity. The main principle of WAL is to apply the requested changes to data files only after they have been logged, i.e., recorded into a log file and the file has been stored in a persistent storage. If the system crashes, it can be recovered using the log file. Therefore, the WAL method ensures fault tolerance. Moreover, the WAL mechanism helps to optimize the system performance, since only the log file (rather than all the data changes) should be written to the persistent storage to guarantee that a transaction is (eventually) committed.

However, an implementation of a persistent storage, i.e., the guaranteeing that the node containing the log file never crashes, is hard to achieve. To ensure resilience, the proposed mechanism can be combined with the required replication techniques as follows. In a distributed data store consisting of a number of nodes distributed across different physical locations, one of the nodes, called *master*, is

appointed to serve incoming data requests from distributed data store clients and transmit back the outcome of the request, i.e., acknowledge success or failure of a transaction. The remaining nodes, called *standby* or *worker nodes*, contain replicas of the stored data.

Each request is first recorded in the *master log* and then applied to the stored data. After this, an acknowledgement is sent to the client. The standby nodes are constantly monitoring and streaming the master log records into their own logs, before applying them to their persistent data in the same way. Essentially, the standby nodes are continually trying to "catch up" with the master. If the master crashes, one of the standby nodes is appointed to be the master in its stead. At this point, the appointed standby effectively becomes the new master and starts serving all data requests. A graphical representation of the system architecture is shown in Fig 14.6.

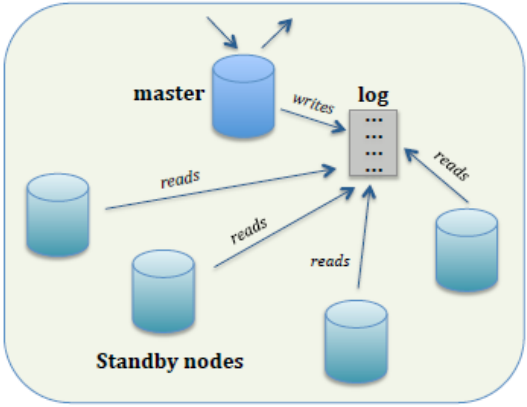


Figure 14.6: Distributed Data Base System Architecture

A distributed data store can implement different models of logging. In the *asynchronous model*, the client request is acknowledged after the master node has performed the required modifications in its persistent storage. The second option - the cascade master-standby - is a *semi-synchronous architecture.* The client receives a response after both the master and its warm standby (called upper standby) has performed the necessary operations. Finally, in the *synchronous* model, only after all replica nodes have written into their persistent storage, i.e., fully

synchronized with the master node, the transaction can be committed. Obviously, such logging models deliver different resilience guarantees in the cases of component crashes.

Modelling Distributed Data Store in *Event*-B. Let us propose a refinement based approach to deriving various system architectures. For each architecture we formulate and prove system-level *logical* properties - *data consistency* and *data integrity*. Within the described system, the *data consistency* properties express the relationships between the requests handled by the master and those handled by the standby nodes. Since any standby node is continuously copying the master log, we can say that any standby node is logically "behind" the master node. The degree of consistency depends on the chosen architecture.

Within the described system, the *data integrity* property ensures that the corresponding log elements of any two storages (master or standby replicas) are always the same. In other words, all logs are *consistent* with respect to the log records of the master node. Essentially it means, that different replicas all do the same operations according to the log records.

We rely on the Event-B refinement technique to gradually unfold the system architecture and functionality. This allows us to represent the system components, model their change (both normal and abnormal) as well as introduce a generic mechanism for changing the master node. We also mathematically formulate the data consistency and data integrity properties for different architectural models. Additionally, formal modelling allows us to identify situations, where the desired properties can be violated.

Below, the refinement process is illustrated for the *asynchronous system* architecture. It consists of the abstract model and two refinements as depicted in Fig. 14.7. A brief outline of each step is given as follows:

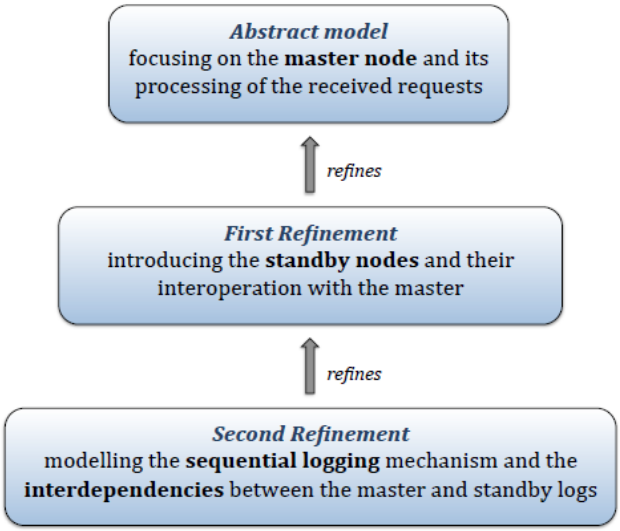


Figure 14.7: Overview of the Development Strategy: Asynchronous model

Initial model. It abstractly represents the overall system architecture. In particular, its describes the behavior of the master node, which is responsible for receiving and processing of incoming requests. Moreover, here we model a possible change in the set of active nodes and introduce an abstract representation of the procedure of a new master selection.

First refinement. This is a refinement of the abstract specification. Here we introduce the behavior of the standby nodes and their interactions with the master. We model how the received data requests are transferred through the different processing stages on the master and standby sides. Moreover, we explicitly model possible node failures, and therefore elaborate on the procedure of selection of new master. At this step we are able to formulate the data consistency properties expressing the relationships between the requests handled by the master and those by the standby nodes, respectively. A short transitional period may be needed for the new master to “catch up” with some of the standby nodes that got ahead by handling the requests still not

committed by the new master. To address this problem, we introduce an explicit representation of the transition period and redefine the consistency property.

Second refinement. This model explicitly introduces the sequential logging mechanism and the resulting interdependencies between the master and standby logs. The model is obtained as a result of a data refinement. An introduction of the sequential representation of the component log allows us to refine some proven invariants as well as prove some new ones. In particular, we formulate and prove the log data integrity properties as the following model invariants:

$$\forall c1, c2, i \cdot c1 \in \text{comp} \wedge c2 \in \text{comp} \wedge \text{i} \in 1\,..\,\text{index\_written(cV)} \wedge \text{i} \in 1\,..\,\text{index\_written(c2)} \Rightarrow log(c1)(i) = log(c2)(i).$$

The property states that the corresponding log elements of any two storages are always the same.

The formal development of the *semi-synchronous* and *synchronous* architectures is essentially repeats the refinement steps presented for the *asynchronous* model. However, in the *semi-synchronous* case, in the abstract model we also introduce the upper standby component and its interoperations with the master node. In both cases, we implement specific architectural solutions for the corresponding architecture and respective restrictions on the component behaviors. Therefore, the data consistency properties for each architecture are reformulated and proved. In its turn, the data integrity property is architectural-independent and remains the same. The resulting Event-B formal models can be served as a starting point for future development of a specific distributed application.

The outlined refinement process supports qualitative reasoning about resilience. However, it is also desirable to quantitatively assess sensitivity of the architecture to changes of its configuration parameters. To enable such quantitative assessment of resilience characteristics, in particular, to analyze the performance/fault tolerance ratio of the architectural alternatives, we integrate formal modelling in Event-B with discrete-event simulation in SimPy. Next we overview the proposed integrated approach.

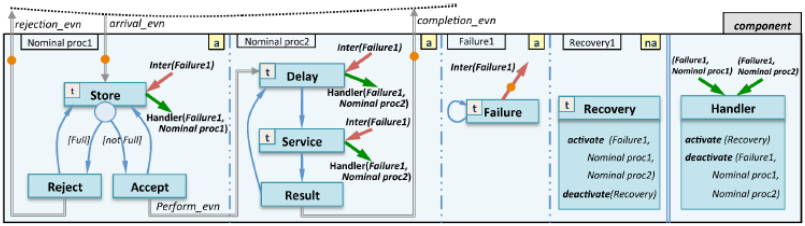
Quantitative Assessment of Resilient Characteristics. To facilitate integration of the described formal modelling with discrete-event simulation, we introduce an intermediate graphical notation called a *process-oriented model*. The proposed notation contains only the main concepts of the domain together with the key artefacts required for both formal modelling and simulation. It relies on the following assumptions:

- A system consists of a number of parallel processes, interacting asynchronously by means of discrete events;
- System processes can be grouped together into a number of components;
- Within a process, execution follows the pre-defined scenario expressed in terms of functional blocks (activities) and transitions between them. Each such functional block is typically associated with particular incoming events the process reacts to and/or outgoing events it produces;
- A system component can fail and (in some cases) recover. The component failures and recovery mechanisms are described as special component processes simulating different types of failures and recovery procedures of the component;
- Some events (e.g., component failures) should be reacted on immediately upon their occurrence, thus interrupting the process current activities. Such special events *(interrupts*) are explicitly described in the component description.

An example of such a component is graphically presented on Fig.14.8. The component interface consists of one incoming event *(arrivaLevn*) and two outgoing events *(rejection_evn* and *completion_evn*). The component itself contains two processes describing its "nominal" behaviour: the first one stores requests to perform a certain service, and the second one performs a requested service and returns the produced results. The internal event *perform^evn* triggers the request execution by the second process. In addition, the component includes the processes Failure and Recovery to simulate possible component failures and its recovery.

Integration Formal Modelling with Simulation in SimPy. A process- oriented model serves as a basis for both Event-B development

and system simulation in SimPy. Translating a process-oriented model into Event-B gives us the starting point for formal development with the already fixed system architecture and the control flow between main system components. The corresponding system properties are explicitly formulated and proved as system invariants.



While translating a process-oriented model to SimPy, we augment the resulting code with concrete values for its basic quantitative characteristics, such as data arrival, service, and failure rates.

Figure 14.8: Example of a system component

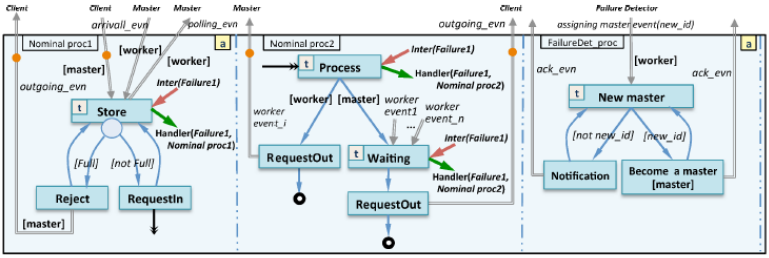


Figure 14.9: Synchronous model

This allows us to compare the system performance and reliability for different system parameter configurations. If satisfactory configuration values can be found and thus re-design of the base process-oriented model is not needed, the simulation results does not affect the Event-B formal development and can be considered completely complementary to it.

We apply the proposed approach to evaluate architectural alternatives combining WAL and replication. We consider two different system architectures: asynchronous and synchronous models. The

resulting process- oriented models for the node components of synchronous architecture is presented in Fig. 14.9.

The graphical notation facilitates development of SimPy code. Discrete event simulation in SimPy allows us to evaluate how different parameters affect the results within the considered architecture.

Fig.14.10 and Fig.14.11 show the results of a simulation involving two models - asynchronous and synchronous. With identical operating conditions and parameters, the *asynchronous* model has higher throughput, completing 99.3 % of requests in 1 hour. This is expected, because the *asynchronous* model has shorter delay in comming transactions than the *synchronous* one, which completes 97.2 % of requests in 1 hour (see Table 14.1).

Table 14.1: Results from model comparison

	Completed (%)	Rejected (%)	Failed (%)
asynchronous	99.3	0	0.2
synchronous	97.2	1.6	0.7

Moreover, for each architecture, we can perform sensitivity analysis. Specifically, we can evaluate the impact of the buffer capacity and the mean failure rate on the throughput of the system. Further experiments can reveal more information about the system. For example, we can evaluate how changing the number of standby node affects the performance of the models and the mean failure rates. In general, the desirable properties and characteristics to be assessed are identified according to the system goals.

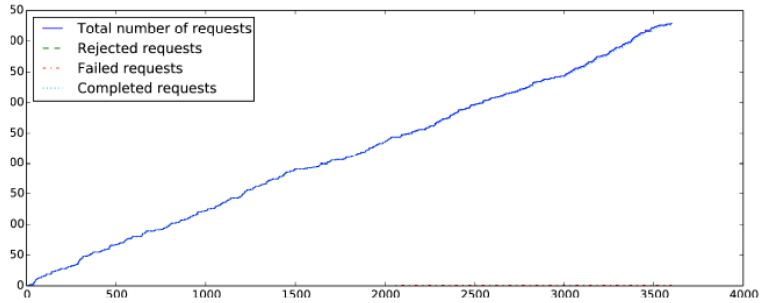


Figure 14.10: Asynchronous model. Mean arrival rate is 7:5=min, service time is 5s, bu_er capacity is 5 and mean failure rate is 1:8=h.

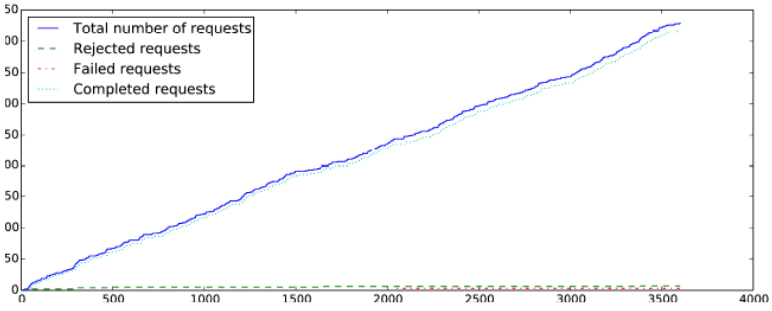


Figure 14.11: Synchronous model. Mean arrival rate is 7:5=min, service time is 5s, bu_er capacity is 5 and mean failure rate is 1:8=h.

To summarize the results of this section, we can conclude that our pragmatic approach to integrating formal modelling in Event-B and discrete-event simulation in SimPy offers a scalable solution to integrated engineering of resilient architectures. Modelling in the Event-B framework allows us to reason about correctness and data integrity properties of the corresponding architectures, while discrete-event simulation in SimPy enables quantitative assessment of performance and reliability.

Advancement questions

1. What are principles the proposed approach relies on?
2. What is the main idea of resilience-explicit refinement process for the systems that perform a certain predefined scenario?
3. What we should implement to model functional decomposition?
4. State the purpose of the Component Modelling and Abstract Reconfiguration Modelling.
5. What for the multi-agent modelling paradigm is used?
6. Explain the idea of Pattern-Based Formal Development of Resilient MAS.
7. How we can formulate the Goal Modelling Pattern?
8. What is the purpose of using Agent Modelling Pattern and Agent Refinement Pattern
9. What is the WAL mechanism and what is the purpose of its implementation?
10. what are main the stages of the refinement process for the asynchronous system architecture?

REFERENCES

[1] Pereverzeva Inna Formal Development of Resilient Distributed Systems / Inna Pereverzeva // PhD diss., Turku Centre for Computer Science, Abo Akademi University, Faculty of Science and Engineering, Joukahaisenkatu, Turku, Finland. – 2015.

[2] Helm R. Elements of Reusable Object-Oriented Software/R.Helm R.EJohnson. J. Vlissides Gamma, E. Design Patterns //Addison-Wesley Reading.-1995.

[3] Hoang T. S. Event-b patterns and their tool support /T. S. Hoang, A. Fu¨rst, and J.-R. Abrial.// Software and System Modeling/-2013.-12(2).-229–244,

[4] Iliasov A. Patterns for refinement automation/A. Iliasov, E. Troubitsyna, L. Laibinis, and A. Romanovsky.// In Formal Methods for Components and Objects - 8th International Symposium : Springer.-2010.-pages 70–88

[5] Tarasyuk A. Integrating stochastic reasoning into event-b development./A. Tarasyuk, E. Troubitsyna, and L. Laibinis //Formal Asp. Comput.-2015.-27(1).-53– 77

V. Sklyar, O. Illiashenko, V. Kharchenko, N. Zagorodna, R. Kozak, O. Vambol, S. Lysenko, D. Medzatyi, O. Pomorova

SECURE AND RESILIENT COMPUTING FOR INDUSTRY AND HUMAN DOMAINS.

Fundamentals of security and resilient computing

Multi-book, Volume 1

Editor Vyacheslav Kharchenko

National Aerospace University n. a. N. E. Zhukovsky
"Kharkiv Aviation Institute"
17 Chkalova street, Kharkiv, 61070, Ukraine
http://www.khai.edu