Lecture Notes in Computer Science 5111

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Qingfeng Chen Chengqi Zhang Shichao Zhang

Secure Transaction Protocol Analysis

Models and Applications

Springer

Authors

Qingfeng Chen
Deakin University
School of Engineering and Information Technology
Melbourne, Australia
E-mail: qifengch@deakin.edu.au

Chengqi Zhang
University of Technology, Sydney
Faculty of Engineering and Information Technology
Centre for Quantum Computation and Intelligent Systems
Sydney, Australia
E-mail: chengqi@it.uts.edu.au

Shichao Zhang
Guangxi Normal University
College of CS and IT, Guilin, China
and
University of Technology, Sydney
Faculty of Engineering and Information Technology
Sydney, Australia
E-mail: zhangsc@mailbox.gxnu.edu.cn

Library of Congress Control Number: 2008931701

CR Subject Classification (1998): C.2, D.1.3, F.1.2, I.2.8, D.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-85073-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-85073-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12327997 06/3180 5 4 3 2 1 0

Preface

The application of formal methods to security protocol analysis has attracted increasing attention in the past two decades, and recently has been showing signs of new maturity and consolidation. The development of these formal methods is motivated by the hostile nature of some aspects of the network and the persistent efforts of intruders, and has been widely discussed among researchers in this field. Contributions to the investigation of novel and efficient ideas and techniques have been made through some important conferences and journals, such as *ESORICS*, *CSFW* and *ACM Transactions in Computer Systems*. Thus, formal methods have played an important role in a variety of applications such as discrete system analysis for cryptographic protocols, belief logics and state exploration tools. A complicated security protocol can be abstracted as a manipulation of symbols and structures composed by symbols. The analysis of e-commerce (electronic commerce) protocols is a particular case of such symbol systems.

There have been considerable efforts in developing a number of tools for ensuring the security of protocols, both specialized and general-purpose, such as belief logic and process algebras. The application of formal methods starts with the analysis of key-distribution protocols for communication between two principals at an early stage. With the performance of transactions becoming more and more dependent on computer networks, and cryptography becoming more widely deployed, the type of application becomes more varied and complicated. The emerging complex network-based transactions such as financial transactions and secure group communication have not only brought innovations to the current business practice, but they also pose a big challenge to protect the information transmitted over the open network from malicious attacks. However, there has been no specialized monograph to consider these issues. Thus this book takes these interesting topics into account and offers innovative techniques for modelling e-commerce protocol, analyzing transaction data and testing the protocol performance in an intuitive way.

The present volume arose from a need for a comprehensive collection presenting the state of the art in security protocol analysis, and is aimed at serving as an overall course-aid and self-study material for researchers and students in formal methods theory and applications in e-commerce, data analysis and data mining. However, the volume can be useful to anyone else who is interested in secure e-commerce.

This book is organized into eight chapters that cover the main approaches and tools in formal methods for security protocol analysis. Having in mind that the book is also addressed to students, the contributors present the main results and techniques in an easily accessed and understood way together with many references and examples.

Chapter 1 is an introductory chapter that presents the fundamentals and background knowledge with respect to formal methods and security protocol analysis. Chapter 2 provides an overview of related work in this area, including basic concepts and terminology. Chapters 3 and 4 show a logical framework and a model checker especially for analyzing secure transaction protocols. Chapter 5 explains how to deal with uncertainty issues in secure messages, including inconsistent messages and conflicting beliefs in messages. Chapter 6 integrates data mining with security protocol analysis, and Chap. 7 develops a new technique for detecting collusion attack in security protocols. Chapter 8 presents a summary of the chapters and gives a brief discussion of some emerging issues.

Although it is not easy to cover all the relevant studies in this book, due to varied formal methods and increasingly complicated security protocols, we hope that it is comprehensive enough to provide a useful and handy guide for both beginners and experienced researchers.

We would like to express our sincere thanks to all colleagues who provided us with useful comments and support during our writing of this book. These include Yi-Ping Phone Chen and Zili Zhang from Deakin University, Li Liu from the University of Technology Sydney, Jeffrey Xu Yu from the Chinese University of Hong Kong, Shuo Bai from the Institute of Computing Technology of the Chinese Academy of Sciences, Xiaowei Yan from Guangxi Normal University and Kaile Su from Sun Yat-Sen University.

We also wish to especially thank Alfred Hofmann, Editor at Springer, for his enthusiasm, patience and great efforts in publishing this book, as well as his staff for their conscientious efforts of providing materials. We are very grateful to all of the reviewers for their useful and valuable feedback. We also thank our families for their persistent support throughout this project.

This work was partially supported by an Australian large ARC grant (DP0667060), a China NSF major research Program (60496327), a National Natural Science Fund for Distinguished Young Scholars of China under

Grant No. 60625204, a China NSF grant (90718020), a China 973 Program (2008CB317108), an Overseas-Returning High-level Talent Research Program of China Ministry of Personnel, and Guangxi NSF grants.

April 2008

Qingfeng Chen
Chengqi Zhang
Shichao Zhang

Contents

1 Introduction 1
1.1 What Is Security Protocol? 1
1.2 Needs of Formal Analysis for Secure Transaction Protocols . 3
1.3 Formal Methods and Related Areas 5
1.4 Emerging Issues and Trends 10
1.5 A Brief Discussion on the Chapters 12
1.6 Summary 13

2 Overview of Security Protocol Analysis 17
2.1 The Formalism 17
2.1.1 Basic Notations and Terminology 18
2.1.2 Inference Rules 19
2.2 Security Protocols 21
2.2.1 SET Protocol 22
2.2.2 Netbill Protocol 23
2.2.3 Security Services 24
2.2.4 Principles of Cryptography 32
2.2.5 Threats in Security Protocols 38
2.3 Research into Analysis of Security Protocols 42
2.3.1 A Discussion of Formal Methods and Security Protocols 42
2.3.2 A Brief Introduction to Protocol Abstraction 44
2.3.3 A Classification of Approaches for Protocol Analysis . 47
2.4 Attack-Construction Approach 52
2.4.1 Approaches by Dolev and Yao 52
2.4.2 NRL Protocol Analyser 56
2.5 Inference-Construction Approach 61
2.5.1 BAN Logic 61
2.5.2 Extensions to BAN Logic 64
2.6 Proof-Construction Approach 67
2.7 Approaches Using Formal Tools and Specification Languages 68

2.8 Summary 71

3 Formal Analysis of Secure Transaction Protocols 73
3.1 Introduction 73
3.2 Research into Verifying Electronic Transaction Protocols . . . 75
3.2.1 Formalism for Protocol Analysis Using Process Calculi 75
3.2.2 Formal Analysis Using an Observational Transition System 78
3.2.3 Formal Analysis of Card-Based Payment Systems in Mobile Devices 80
3.3 A Computational Model 83
3.4 Basic Terms and Statements 86
3.5 Logical Framework and Statement of ENDL 89
3.5.1 Axiom 90
3.5.2 Inference Rules 94
3.5.3 Inference Format 99
3.5.4 Verification Instances of Security Protocols in ENDL . 99
3.6 Summary 106

4 Model Checking in Security Protocol Analysis 107
4.1 An Overview of Model Checking in Analysing E-Commerce Protocols 108
4.1.1 Model Checking for Failure Analysis of Protocols 109
4.1.2 Automatic Analysis of E-commerce Protocols Using UML 111
4.2 An ENDL-Based Verification Model 113
4.2.1 Components 113
4.2.2 Designing the Model 114
4.2.3 Handling the Knowledge and Facts 117
4.2.4 Recognition 118
4.3 Comparison with Theorem Proving 125
4.4 Discussion 127
4.5 Summary 129

5 Uncertainty Issues in Secure Messages 131
5.1 Introduction 131
5.2 Estimation of Inconsistency of Secure Messages 134
5.2.1 Related Work 134
5.2.2 Semantics Description 137
5.2.3 Measuring Inconsistency in Secure Messages 144
5.2.4 Examples of Measuring Inconsistency 151
5.2.5 Experiments 153
5.3 Integration of Conflicting Beliefs in Secure Messages 156
5.3.1 Related Work 157

5.3.2 Basic Concepts . . . 159
5.3.3 Handling Inconsistent Beliefs in Secure Messages . . . 165
5.3.4 Experiments . . . 170
5.4 Summary . . . 172

6 Applications of Data Mining in Protocol Analysis . . . 175
6.1 Introduction . . . 175
6.2 Related Work . . . 177
6.3 Basic Concepts . . . 180
6.4 Association Rule Mining for Inconsistent Secure Messages . . 182
6.4.1 The Basics of Association Rule Mining . . . 182
6.4.2 Data Preparation . . . 184
6.4.3 Identifying Association Rules of Interest . . . 186
6.5 Algorithms and Experiments . . . 188
6.5.1 Algorithms . . . 188
6.5.2 Experiments . . . 189
6.6 Summary . . . 192

7 Detection Models of Collusion Attacks . . . 193
7.1 Introduction . . . 193
7.2 Related Work . . . 195
7.3 Identification of Frequent Patterns for Collusion Attack Detection . . . 198
7.3.1 Basic Concepts . . . 198
7.3.2 A Framework to Detect Collusion Attacks . . . 200
7.3.3 Dealing with Knowledge and Facts . . . 202
7.3.4 A Case Study . . . 203
7.4 Estimation of the Probability of Collusion Attacks . . . 205
7.4.1 Motivations . . . 205
7.4.2 Preliminaries . . . 206
7.4.3 Identifying Collusion Attack Using Bayesian Network 208
7.4.4 Experiments . . . 212
7.5 Summary . . . 215

8 Conclusion and Future Works . . . 217
8.1 Conclusion . . . 217
8.2 Future Work . . . 219

References . . . 223

Index . . . 233

1

Introduction

Security protocols (cryptographic protocol) have been widely used to not only achieve traditional goals of data confidentiality, integrity and authentication, but also secure a variety of other desired characteristics of computer-mediated transactions recently. To guarantee reliable protocols, a great deal of formal methods has been undertaken not only to develop diverse tools with specialized purpose or general purpose, but also to apply them to the analysis of realistic protocols. Many of them have been proved to be useful in detecting some intuitive attacks in security protocols. In many cases, a useful feedback is supplied to designers in order to improve the protocol's security. For both beginners and experienced researchers, this book will present useful information on relevant technologies that can be extended or adapted. A comprehensive introduction to the basic concepts and core techniques will be presented. In this chapter, we explain what is security protocols and how they can be used to ensure secure transactions, what challenging issues in e-commerce (electronic commerce) are, why security protocol analysis important, how they are performed, and what are the ongoing efforts and relevant work. We will also explain the limitations in previous work and why it is important to develop new approaches. These questions will be briefly answered. In particular, we will focus on the discussion regarding secure transaction protocols. Finally, some emerging issues and the ways they are being met are also described.

1.1 What Is Security Protocol?

First, let us consider a financial transaction that is to send sensitive data such as *Alice*'s credit card numbers to a vendor like Dell Inc. Several occurring matters in this transaction are listed below.

- credit card number, ID
- encrypted credit card number, no ID
- encrypted credit card number, ID

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 1–15, 2008.

The first case provides no encryption protection to the credit card number. An intruder can see the credit card number and masquerade as *Alice* to proceed the transaction using the ID. The credit card number is encrypted in the second case, whereas the vendor cannot authenticate the sender's identifier that might be missing, intercepted or tampered by malicious hackers. Only the last case may be safe since it encodes the credit card number and includes the ID.

In the past few years, researchers have sought to develop techniques for information security. One of the most effective and popular ways is the application of security protocols. A security protocol is a sequence of operations that perform a security-related function by using cryptographic methods. A protocol specifies how the cryptography should be used, and includes details about data structures and representations. For example, it is not easy for the intruder to see *Alice*'s credit card number without the right key.

There are a variety of protocols for different purposes, such as communication protocols. File transfer protocol (FTP) that is a protocol to describe file transfers between a host and a remote computer; hypertext transfer protocol (HTTP) is the set of rules for exchanging files (text, audio, video, and other multimedia files) on the World Wide Web; and electronic transaction protocols for secure e-commerce. Usually, a security protocol has to incorporate some of the following aspects to ensure secure data transport.

- *Entity authentication.* This means the authentication of principals, by which to ensure users are who they say they are. One familiar example is access control. A computer system supposed to be used only by those authorized must attempt to detect and exclude the unauthorized.
- *Key agreement or establishment.* This is to make two or more parties agree on a session key.
- *Encryption construction.* This is the process of converting information to make it unreadable without special knowledge. It has been widely used to protect communications. Although encryption can be used to ensure secrecy, other techniques are still needed to verify the integrity and authenticity of a messages.
- *Secure data transport.* This provides secure communication on the distributed systems in combination with various cryptographic mechanisms, such as public-key (asymmetric) cryptography, symmetric ciphers, one-way hash functions and so on. Furthermore, some new devices like timestamps and key-sharing are also used recently. We will give explanation to the above concepts in the next section.

Secure transaction protocol (e-commerce protocol) [55, 140] is one of the important security protocols, and has been mainly developed to secure financial transactions. It specifies transaction rules that must be conformed in each processing phase. To complete a transaction, for example, *Alice* needs to ob-

tain a valid credit card number from a authorized financial institute, have the correct PIN to access the credit card, sends the encrypted credit card number along with relevant information such as identifier to the vendor; and the vendor must decrypt the message and send a response message to *Alice* to confirm the transaction.

With the rapid growth of online trading, the reliability of e-commerce protocols has received a great deal of attention. This book aims to present some innovative techniques for secure transaction protocol analysis. It considers the characteristic of financial transactions and focuses on building models for examining and evaluating the protocol performance using transaction data.

1.2 Needs of Formal Analysis for Secure Transaction Protocols

Most internet users may have experiences of buying products online such as shares, computers or foods, or transferring money by using internet banking. It is natural that they might be concerned about revealing their credit card numbers, personal details, or receiving wrong products. In other words, people wonder the transaction may be unsecured. The development of formal methods owes much to the security community. A number of formal security models, tools for reasoning about security, and applications of these tools to proving systems secure were developed in the 1970s and early 1980s. The wide use of the internet brings these security problems to the attention of the masses.

The emergence of e-commerce has caused innovation in current business practice, and has broken through conventional marketing barriers, as activities on the internet are no longer limited to time and geography. Unlike conventional business, the development of e-commerce is unprecedented. There has been a vast growth in retail e-commerce and in transactional use by small business. The following industry forecast should be sufficient to indicate the dynamic growth and potential of electronic commerce:

> *Forrester forecasts that the world total e-commerce (B2B and B2C) has been expected to reach 2.3 trillion by 2002 and to be on track to reach 13 trillion by 2006. The compound annual growth rate is around 53.6 per cent* [73].

Furthermore, e-commerce improves the efficiency of existing business models and enables the transformation of these models, which present reduced costs and increasing competitiveness, as well as bring new challenges. This has resulted in the development of a great many e-commerce systems. With the development of e-commerce systems, their security has become a key issue. For example, people may hesitate to send their credit card number or date of birth

to an electronic transaction system when asked for it on-line. The vendor must be able to provide adequate protection from fraud and violation of privacy when trading on the Web. Currently, consumers can find a great variety of systems offered by vendors on the Internet, but secure payment capability has not been guaranteed. It is thus a high-profile problem for e-commerce systems to protect the information transmitted over the open network from malicious attacks.

In general, security in e-commerce is implemented by relying on a set of secure protocols that meet the user's expectation for secure transactions. However, existing security protocols are not always secure enough to meet people's expectation. Besides, the design of a security protocol is difficult and error-prone. In particular, some subtle flaws have been recognized in popular and widely-used security protocols. This generates a crucial requirement of identifying weakness and hidden flaws in security protocols.

The traditional ways of verifying security protocols are through human inspection, simulation, and testing. Unfortunately these approaches provide no guarantees about the quality of security protocols. Formal methods comprise a variety of mathematical modelling techniques for specifying and modelling the behaviour of a system, and may mathematically verify that the system design and implementation satisfy system functional and safety properties. One of the main applications of formal methods is to assist in security protocol analysis. Formal methods allow us to

- Specify the system's boundary: the interface between the system and its environment.
- Characterize a system's behaviour more precisely in handling both functional behaviour and real-time behaviour by a mathematical or logical model.
- Provide precise definition for the system's desired properties by formulating its requirements.
- Implement a thorough analysis of different paths which an intruder can utilize.
- Prove a system meets its specification by rigorous proofs. Some methods may offer counterexamples if it is not the case.

Clarke and Wing capture the three threads in the development of formal methods in their paper for 1996 *ACM Computing Surveys* [37]-model checking, theorem proving, and software specification. Model checking, in particular, is a proven success for hardware verification; companies such as Intel are establishing their own hardware verification groups, building their own verification systems, and hiring people trained in formal methods.

As more and more transactions are carried out via computer networks, however, and as cryptography is widely applied, the types of applications in which the security protocols need to be integrated becomes more varied and

complex. As Meadows indicated [112], some emerging issues and trends with respect to formal analysis for security protocols require us to develop new approaches that must take these issues into account. Most previous approaches have concentrated on formulating security problems in terms of the properties of a discrete problem such as intercepting data, concatenating and deconcatenating data, and encrypting and decrypting data. The field has now reached a state in which there are many different tools available that can be used to verify various security properties of security protocols.

Some recent trends in security protocols present new challenges to protocol analysis. As computer networks become more widespread, and more transactions are handled in a potentially hostile environment, the security protocols become more complex and varied. This makes protocol analysis more difficult. Most existing protocol analysis systems use a very simple model of encryption but cannot handle the modelling of properties possessing many current algorithms. On the other hand, some new types of threats such as denial of service and traffic analysis, and new applications such as financial transactions, require us to take these new issues into account when attempting to develop approaches to assure correctness of security protocols.

1.3 Formal Methods and Related Areas

Formal methods have been widely used in security protocol analysis since the 1980s. Regardless of the various classifications of formal methods in the relevant literature, formal methods are classified into theorem proving, model checking and formal logics in this book according to the usual definition.

Theorem proving uses higher-order logic to reason about possible protocol executions by constituting a compelling proof that a particular property always holds. These logics are not subject to finite bounds, and provide mechanized proofs, including automated tools and proof checking, which can assist in parts of proofs and prevent errors in reasoning.

An inductive proof starts with defining a set of traces. Given a protocol, a trace is one possible sequence of events, such as attacks. It aims to prove correction of protocols by induction and confirm whether a particular description of a protocol meets a particular property. In other words, for every state in every trace, it is able to prove that no security condition fails. This is true even when the system can engage in arbitrarily interleaved runs of the protocol, when the space of facts is infinite, and when the number of users is unbounded. To check with the secrecy and authentication properties, it focuses on the fact that certain messages should not occur, or should occur only in specific situations. Usually, we are more concerned with offering theories for establishing the impossibility of a collection of particular events.

The counter-example form of induction is widely used in generating counter-examples. In this case, a theorem prover may be terminated when any valid theorem can eventually be proven.

Alternatively, they may find a counter-example in the process. Various proof search strategies are used, often based on the basic resolution strategy proposed by Robinson [133]. Inductive theorem proving, as in [111], can be general, but requires time-consuming interactive theorem proving by experienced researchers.

The following four safety properties are all goals needed to be achieved in all security protocols.

- *Authenticity* is to confirm who sent a message. If principal A receives a message from B but thinks it is from C, the proof fails.
- *Integrity* is to check whether a message has been altered. If A receives a message from B but this message is different from what B sent, the proof fails.
- *Secrecy* is to ensure that only the valid principals can receive the message. If an intruder knows a message that should be kept secret from him/her, the proof fails.
- *Anonymity* is to make the principals' actions unknown from the intruders. If an intruder or principal B knows an action is performed by A, the proof fails.

Great efforts have been devoted to developing induction approaches for security protocol analysis. One remarkable work is Isabelle [124] theorem prover by Paulson. It supports proof development including high-order logic and generic treatment of inference rules.

Model checking is a method to formally verify a finite-state concurrent system. The specification of the system is often written as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system. The verification is achieved by checking if the formal specification can be satisfied by the model.

The model is usually expressed as a transition system, i.e a directed graph consisting of nodes (or vertices) and edges. A collection of atomic propositions is associated with each node. The nodes denote states of a system, the edges denote possible executions which alter the state, while the atomic propositions denote the basic properties that hold at a point of execution.

Formally, a problem can be stated in terms of the following steps: given a desired property (*modelling*), expressed as a temporal logic formula p (*specification*), and a model M with initial state s, decide if $\boldsymbol{M\ s} \models \boldsymbol{p}$ (*verification*). Modelling converts the system into a formalism using a state transition graph such as *Kripke* structure. It may sometimes require the use of abstraction due to the limitations of time and memory. For example, a Kripke structure can

be represented by a set of atomic propositions (AP), which is quadruple, $M = (S, S0, R, L)$ where

- S represents a finite set of states.
- $S0$ represents the set of initial states, $S0 \subseteq S$.
- $R \subseteq S \times S$ represents a transition relation.
- L: $S \rightarrow \{AP\}$ represents a function that labels each state with the set of atomic propositions in this state.

Specification uses some operators in temporal logic to describe the system. For example, X for 'next time', F for 'in the future', G for 'globally', U for 'until', and R for 'release'. Also two quantifiers can be used, including A for 'always' and E for 'exists'. There are two main specification languages to represent temporal logic, including **CTL** (Computation Tree Logic) and **LTL** (Linear Temporal Logic).

The verification aims to check whether the final system satisfies an expected property (called specification). The model checking tool outputs *yes* if the given model satisfies the given specification and generates a counterexample otherwise. The counterexample explains why the model does not satisfy the specification. By studying the counterexample, we can know the source of the error, which is able to provide useful information for us to find a solution to the problem. Nevertheless, model checking tools face the state explosion problem. A number of studies such as symbolic algorithms, partial order reduction, abstraction and on the fly model checking have been developed in order to cope with this problem.

Recently, many state exploration tools based on the Dolev-Yao model have been developed for security protocol analysis owing to Lowe's demonstration that it was possible to use a general-purpose model checker, FDR, to detect a man-in-the-middle attack on the Needham-Schroeder public key protocol [97]. Most of the succeeding work applied both model checking and theorem proving to the problem, as well as in the design of special-purpose model checkers. On the other hand, there have also been studies showing under what circumstances checking a finite number of states might be sufficient.

Formal logic is one of the critical tools in formal methods. For example, temporal logics play an important role in model checking. Formal logic focuses on the study of inference with a set of rules for making deductions that are made explicit. It formalizes such deductions with precise rules to decide if an argument is valid. This can be achieved by representing objects and relationships symbolically including the quantifiers and logical connectives such as $\exists$, $\forall$, $\vee$, $\wedge$, $\rightharpoonup$, and $\rightarrow$.

Formal logic encompasses a wide variety of logical systems, such as term logic, predicate logic and modal logic, and formal systems are indispensable in all branches of formal methods. The Burrows, Abadi and Needham (BAN) [22] logic is a representative belief logic, which consists of a set of modal operators

describing the relationship of principals to data, a set of possible beliefs about important components (such as a belief that a message was encrypted by a certain principal's public key) and properties (such as freshness that a message is not reused or replayed) of protocols that can be hold by principals, and a set of inference rules that define the semantics of BAN logic and is able to derive new beliefs from old ones. An example would be a rule saying that if A believes that a message m is fresh, and that B once said m, then A believes that B believes m in the current run of the protocol. BAN logic provided an intuitive and simple set of rules, and was successfully used to detect subtle flaws in protocols [2]. Consequently, the logic results in a number of other logics that extend or adapt it to different types of problems in cryptographic protocols. Although belief logics are usually weaker in comparison with state exploration tools owing to a much higher level of abstraction, they are decidable, efficiently computable, and even completely automated [18].

Temporal logic represents and reasons about temporal information within a logical framework, and is usually used to define the semantics of temporal expressions in natural language. Consider the statement: 'A sends a message m to B'. Although its literal meaning is constant in time, the truth value that can vary in time. Sometimes the statement is true, and sometimes the statement is false. In a temporal logic, statements can have a truth value which can vary in time. The three basic temporal operators are always, sometimes, and never. Computational tree logic (CTL), Linear temporal logic (LTL) and Interval temporal logic (ITL) are examples of temporal logics. Computational tree logic (CTL) is a temporal logic. It is often used to express properties of a system in the context of formal verification or model checking. CTL uses atomic propositions to make statements about the states of a system, which are then combined into formulas using logical operators and temporal operators. LTL logic is a modal temporal logic with modalities referring to time, by which one can encode formulae about the future of paths by propositional variables, logical connectives, and temporal modal operators. ITL logic is a temporal logic for representing both propositional and first-order logical reasoning about periods of time, which is capable of handling both sequential and parallel composition.

There are a number of international computer security conferences and journals that publish high quality papers in security areas. A brief list (in our opinion only) is described below for your reference in terms of the accepted paper quality and impact.

Top-ranked conferences:

- **IEEE Symposium on Security and Privacy**. Since 1980, it has been the premier forum for the presentation of developments in computer security and electronic privacy, and for bringing together researchers and practitioners in the field. Papers offer novel research contributions in any

aspect of computer security or electronic privacy. Papers include advances in the theory, design, implementation, analysis, or empirical evaluation of secure systems, either for general use or for specific application domains.

- **ACM Conference on Computer and Communications Security**. Since 1993, the conference seeks submissions from academia and industry presenting novel research on all theoretical and practical aspects of computer security, as well as case studies and implementation experiences. Practical papers have practical relevance to the construction, evaluation, application, or operation of secure systems. Theoretical papers make convincing argument for the practical significance of the results.
- **International Cryptology Conference**. Since 1995, the International Association for Cryptologic Research has sponsored the annual Cryptologic Research (Crypto) and European Cryptologic Research(Eurocrypt) conferences. Since 2000, it has also sponsored the annual Asian Cryptologic Research (Asiacrypt) conference. Their purpose is to further research in cryptology and related fields.
- **USENIX Security Symposium**. Since 1993, the conference brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security of computer systems and networks. It includes the papers covering novel and scientifically significant practical works in security or applied cryptography.

Top-ranked journals:

- **ACM Transactions on Information and System Security**. It includes three top of interests. *Security Technologies* such as authentication; authorization models and mechanisms; auditing and intrusion detection; cryptographic algorithms, protocols, services, and infrastructure; cryptanalysis and formal methods; and secure systems. *Security Applications* such as threats, system tradeoffs, representative application areas include information systems, workflow, electronic commerce, and telecommunications systems. *Security Policies* such as confidentiality, integrity, availability and privacy policies.
- **Computers & Security**. It aims to provide a combination of leading edge research developments, innovations and sound practical management advice for the professionals involved with computer security, audit, control and data integrity in all sectors - industry, commerce and academia.
- **Journal of Cryptography**. It is the official journal of the International Association for Cryptologic Research. This journal covers cryptography and cryptanalysis, including information theoretic and complexity theoretic perspectives, and also includes public key and conventional algorithms and their implementations, and computational number theory and cryptographic protocols.

- **Designs, Codes and Cryptography**. There is a great deal of activity in design theory, coding theory and cryptography. This journal provides a forum for high quality papers of both a theoretical and a practical nature which bridge more than one of these subjects.

1.4 Emerging Issues and Trends

As more and more transactions such as financial transactions by e-commerce system are performed electronically on open networks, and as more and more sensitive and confidential messages must be transmitted in some manner, the concerns over information security challenge us and attract more and more attention.

As network-based electronic transactions become more and more prevalent, the types of transaction become more varied and complicated. For example, financial transactions depend on properties such as freshness and fairness as well as security properties protected by traditional security services. These include secure group communication, in which a key must be kept secret as members can join or leave, and negotiation of data structure such as security association rather than keys. Furthermore, the new types of threats, such as collusion attacks, denial of services and traffic analysis, become more implicit and dangerous. We have to be concerned not only about the external threats that may attempt to break the protection of secrets or impersonate the honest principals, but the internal threats that may attempt to gain unauthorized access to the secrets by collusion with a number of dishonest principals. The primary emerging issues and trends are described below.

- **Increasing complexity**: As networks become more widespread and handle more and more transactions in a hostile environment, security protocols play a central role in guaranteeing secure transactions. The properties they are supposed to satisfy are diverse in different cases. In a protocol with respect to financial transactions, the liveness and fairness are what it mainly wants to achieve; but in a protocol regarding secure group communication, a key is required to remain secret within a group no matter how many members may join or leave. The varied and complex types of application increase the complexity of protocols. For example, the Internet Key Exchange (IKE) protocol needs not only to set up a shared session key, but also to specify the algorithms that use this key. Also, the SET protocol must be able to process various payment card transactions over open networks. We have to increase the complexity of protocols to cope with this challenge. It will certainly make the analysis of protocols more difficult. However, faced with these unavoidable facts, we have to meet them when attempting to perform analysis of such security protocols.

- **New types of cryptography**: Ideally, the cryptographic algorithms are assumed to be reliably secure. However, with the increasing computational power, some algorithms that were believed to be secure have been viewed as no longer guaranteeing sufficient security, such as DES. Furthermore, many of the new algorithms include a number of properties that cannot be modelled by existing analysis methods. Many of them use a simple model to abstract the encryption and decryption, authentication, and so forth. However, other algorithms and data structures consists of new desirable properties, such as anonymity in group key management protocols, as in [26], and the timestamp used in TESLA protocol to provide multicast packet authentication.
- **New applications and new threats**: In traditional computer security, the focus has been on hypothetical threats where there would be an obvious profit for the intruders, such as reused or replayed message. With the introduction of new security protocols, a number of new threats have emerged recently. They have properties that are somewhat different from conventional attacks, and cannot be modelled by traditional methods like the Dolev-Yao model. Many security protocols are subject to denial-of-service attacks [112], in which the attackers initiate a protocol run and then cast off it, leaving a responder committing its resources to maintaining the connection with the initiator. This may cause the responder to waste resources in maintaining the communication between them, and eventually using up all of its available resources. Therefore, some approaches have been taken to resolve this problem by comparing the resources expended by an initiator (defender) to the resources expended by an attacker [81].
  Anonymous communication has recently been introduced because users are concerned about the dangers of traffic analysis that may reveal their individual information over the open network. Actually, even when encryption is applied, it is not easy to hide the source and destination of message traffic, from which an attacker can learn much about the communication secrecy using statistical analysis that depends on correlation of data. Thus, many systems, such as Onion Router, are designed to prevent the attacker from determining the source and destination of requests. However, they lack sufficient ability to measure the degrees of protection provided by these systems.
  There are also some tools that combine the formal methods and statistical techniques. Electronic transactions in electronic commerce include new security properties such as fairness and liveness. However, unlike the safety properties in conventional cryptographic protocols, the desirable properties of electronic commerce protocols cannot be expressed directly using tools designed for checking safety properties. Fortunately, we have seen inspiring efforts in this area. Most work on the application of formal methods to security protocol analysis has focused on the explicit attacks

from external but has paid less attention to the potential or hidden attacks from internal sources. A hostile intruder may intercept, modify, or delete messages, and even handle covert attacks in collusion with a certain number of dishonest principals. For example, suppose principals A, B and C know message (m_1), (m_1, m_2) and (m_3), respectively. Usually, they should know their individual messages only. Nevertheless, existing protocols do not consider that A, B and C may collude to disclose the messages to an intruder Z. The attack is covert and is not apparent even to careful inspection. Therefore, the new types of threats are more dangerous than traditional attacks.

1.5 A Brief Discussion on the Chapters

The aforementioned issues owing to the complexity and diversity of security protocols make people become more concerned about the security of electronic transactions. Thus, this book focuses on discussing effective and innovative methods by modelling transaction systems, analysing transaction data and evaluating protocol performance. Chapter 2 gives an overview of other related works, which include the fundamentals of cryptography, the traditional formal methods, security protocols, and existing verification systems. Furthermore, several considerations are listed below.

New properties of financial transactions. The complexity of security protocols makes the analysis of security protocols become a difficult and error-prone task. They consist of a number of extremely subtle properties, such as freshness in a financial transaction. Even a simple concept of encryption may consist of various specific approaches such as RSA and DES. For example, the vendor may want to confirm that the buyers's credit card number is encrypted by a fresh key rather than an expired key. The precise meaning of this notion must be discussed and made clear. Most work on formal analysis of security protocols focused on verifying properties that can be expressed according to conditions on system traces. To express the properties of e-commerce protocols, Chapter 3 described a formal logic, namely ENDL (extended non-monotonic dynamic logic), including the fundamental notations, axioms, theorems and inference rules. Moreover, Chapter 4 presents an ENDL-based verification model, including the inference engine and the knowledge base, to enable an automatic way for protocol analysis. It is usually designed to analyse the approximation of the safety properties.

Uncertainty issues that consist of inconsistent data problem and inconsistent belief problem. **Inconsistent transaction data**. The transaction data transmitted between principals may be inconsistent with each other. For example, *Alice* might send her credit card number to the *vendor*, whereas the vendor may receive a tampered credit card number or receive nothing at

all. Such inconsistency between transaction data of different principals may imply potential flaws of protocols. Chapter 5 presents the inconsistency issues in secure messages and aims to enhance the protocol analysis by measuring the degree of inconsistency. **Inconsistent belief in transmitted messages**. Traditional protocol analysis assumes that the principals and communication channel are trusted but usually fail to model insecurity. For example, *Alice* believes that a key has a 95% probability to be true, and *Bob* believes that it is true in a 75% probability only. Unfortunately, the previous methods cannot model the imperfect working conditions and verifying the protocol in hostile circumstances. Chapter 5 describes the inconsistent belief in secure messages and shows how to integrate conflicting beliefs.

Hidden associations between secure messages. No matter what security flaws occur in security protocols, they may result in inconsistent transaction data. However, the transmitted messages are not independent but correlated with each other. For example, a PIN number is related to a credit card number and an encryption key has relation to its corresponding decryption key. If a set of secure messages are highly inconsistent between principals, the frequency of the itemset should be low. It is reasonable to suspect the protocol performance. Thus, this makes it possible to detect security flaws from protocols using advanced data mining techniques. Chapter 6 shows how to apply data mining techniques to assist in verifying security protocols, in which it presents the conversion between secure messages and items.

Collusion attacks by a group of dishonest principals. An attack cannot be performed by an individual, whereas it may be proceeded by a group of dishonest principals. For example, the intruder can obtain *Alice*'s credit card number and the PIN number from the vendor and the issuer, respectively. Unfortunately, this threat cannot be modelled by current methods. The potential correlations between transmitted secure messages that are shared by multiple principals provide an intuitive way to identify the threat. Chapter 7 proposes a framework to deal with the emerging security threat, namely collusion attack. It enables an intuitive way to detect the collusion attack.

Finally, Chapter 8 concludes the book and presents the authors' directions for future research.

1.6 Summary

In this chapter, we have briefly introduced the fundamental background knowledge that is relevant to the application of formal methods to security protocol analysis. Formal analysis for security protocols has gradually come to a stage of maturity and consolidation after being developed for the past twenty years. However, emerging issues and trends in this field challenge us to find new solutions for them.

A variety of security protocols have been developed to provide information on the standards and rules used for secure communication between the many points within a computer network and across the internet. In particular, e-commerce protocols have recently attracted more attentions. Regardless the rapid growth of online trading, people become more concerned about the safety of electronic transactions due to increasingly complex protocols and hostile environments. This poses more pressure to verify the reliability of e-commerce protocols.

Formal methods can be viewed loosely as a combination of an abstract mathematical model of the behaviour of a system and corresponding requirements regarding the correctness of the system, together with a formal proof to determine the system has all the required properties that together make it functional and useful. Traditional analysis of security protocols depends on human inspection, simulation, and testing. Unfortunately it cannot assure the quality of security protocols.

Formal methods can be classified into theorem-proving, model-checking and formal logic. **Theorem proving** uses higher-order logic to reason about possible protocol executions, and provides mechanized proofs including automated tools and proof checking. Some of them may provide alternative counter-examples. **Model checking** is to verify a formally specified system by checking whether it can be satisfied by the model. The specification is usually realized using temporal logic such as CTL and LTL. In addition to temporal logic, term logic, predicate logic and modal logic are indispensable in all branches of formal methods. There have been a number of publications in the security areas. Several top-tanked international conferences and journals that have high impact factors and publish high quality papers are presented in this chapter for readers' reference.

As network-based electronic transactions become more and more dependent on computer networks, and as cryptography becomes more widely used, the types of transaction to which a cryptography can be applied can become varied and complicated, such as financial transactions that may include new security properties such as fairness and freshness. Also, the types of new threats become more varied, such as denial-of-service and traffic analysis, and pose a new challenges to traditional formal methods for security protocol analysis.

The financial transactions depend on properties such as freshness as well as security properties protected by traditional security services. The latter include secure group communication, in which a key must be kept secret as members can join or leave, and negotiation of data structure such as security association rather than keys. Furthermore, the new types of threats, such as collusion attacks, denial-of-services and traffic analysis, become more implicit and dangerous. We have to be concerned not only about the external threats that may attempt to break the protection over secrets or impersonate the

honest principals, but the internal threats that may attempt to gain unauthorized access to the secrets by collusion with a certain number of dishonest principals.

This chapter also gives a brief discussion to the challenging issues that are focused in this book, including the formal methods especially for the secure transaction protocol analysis, uncertainty issues to be solved, application of data mining in protocol analysis and detection of collusion attacks.

2

Overview of Security Protocol Analysis

This chapter gives an overview of the fundamental concepts and formalism with respect to formal analysis and verification of security protocols that will be used in the rest of the book. Some recent studies into developing new formal methods to cope with the emerging issues and threats in this research field are also introduced to make clear the challenge in front of us and the current situation.

This chapter is organized as follows. In Section 2.1, we start with introducing some basic concepts, including the notations and terminology used in the book. Section 2.2 presents several security protocols, security service provided by protocols and principles of cryptography. In Section 2.3, we introduce the research into analysis and verification of security protocols. These are classified into different categories based on their specific purposes and theoretical foundation. Section 2.4, Section 2.5, Section 2.6 and Section 2.7 describe the attack-construction approach, inference-construction approach, proof-construction approach and other approaches using formal tools and languages, respectively. Finally, we summarize this chapter in Section 2.8.

2.1 The Formalism

A security protocol is usually described by enumerating the messages transmitted between the principals, and by symbolically indicating the source, the destination, and the contents of each message. However, it seems that this conventional notation is not convenient to manipulate by logic, since we wish to integrate exact meanings into each part of each message and the meanings are not always obvious from the data contained in the messages. Thus, it is necessary to introduce a more useful notation, which not only transforms each message into a logical formula but also preserves correspondence with the original specification of the protocols. This logical formula is an idealized version of the original message. For example, in BAN logic [22], each idealized

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 17–72, 2008.

protocol is annotated by assertions, which usually describe beliefs held by the principals.

In the remainder of this section, we give a brief description of the concepts and notations, the postulates (inference rules), the fundamental methods for formal analysis of security protocols, including inference-construction methods, attack-construction methods, proof-construction methods and formal specification languages and tools.

2.1.1 Basic Notations and Terminology

The objects of security protocols can be classified into three categories; principals (also called participants), encryption keys, and formulae (also called statements). Usually, the uppercase symbols C, M, P, A, AS, X, Y, Z, and CA denote specific principals; k ranges over the encryption keys; $Spv()$ and $Kpv()$ denote the private signature key and key-exchange key, respectively, and $Spb()$ and $Kpb()$ denote the corresponding public signature key and key-exchange key, respectively; m indicates the transmitted messages; $CertS()$ and $CertK()$ denote the signature and key-exchange certificates respectively; $H()$ denotes the one way hashing function; α and β represent a sequence of actions; P and Q represent the statements. All these symbols can be used either as metasymbols or as free variables.

The conjunction is the connective, denoted by comma or $\wedge$. It should satisfy the properties of commutativity and associativity. The negative is denoted by $\neg$. Other constructs are defined below.

Send(X, Y, m): X sends the message m to Y.
Generate(X, m): X generates the message m.
$P \longrightarrow Q$: Q can be concluded if P is true. This construct indicates axiom for P can be composed of several formulae with conjunction, such as $P = P_1 \wedge P_2 \wedge \ldots \wedge P_n$, $n \geq 1$.
$P \longleftrightarrow Q$: P is an equivalent proposition of Q.
$\vdash_\alpha Q$: Q will be true if the action α is successfully finished.
$P \vdash_\alpha Q$: Based on the current P and α, we can conclude that Q is true.
$P \mid\sim_\alpha \neg Q$: Based on P and α, we can non-monotonically conclude Q is untrue.
e(m, k): The message m is encrypted by the symmetric key k.
E(m, k): The message m is encrypted by the public key k.
S(m, k): The message m is encrypted by the private key k.
Auth(X, Y, m): X authenticates message m sent by Y.

On the other hand, this section describes some of the terminology used in the following context. Since many researchers define their own terms, it is necessary to standardize them according to the definitions below.

- **Cryptanalysis** is the science of breaking the cipher text without knowledge of the key. The approaches mainly depend on the letter frequency, and any information with respect to the context available. This technique is very complicated and has the potential to defeat all but the best encryption methods.
- **Strong encryption** is an encryption method that is hypothetically believed to be computationally unbreakable. In addition, it is robust and invulnerable to any form of cryptographic analysis.
- **Authentication protocol** can be interchangeably used with *authentication protocols* and *cryptographic protocols*. It consists of a set of rules specifying the messages transmitted in an encryption system and assists in distributing secret keys.
- **Belief logic** is based on belief. The reasoning systems use the rules about how belief is propagated to achieve new beliefs.
- **Epistemic logic** is based on knowledge. Although the reasoning is similar to belief logic, these logics are used to reason about knowledge rather than belief.
- **Session key** is a secret key shared between two communication principals only. As the name implies, this key is established for one session only. It will expire once the session ends.

More treatment of the changes of these constructs can be seen in the following chapters. Next, the inference rules used to characterize them are outlined.

2.1.2 Inference Rules

As mentioned in [22], in the process of authentication, we have to be concerned with the distinction between two periods, the past and the present. The present starts at the beginning of a special transaction in protocols. All messages transmitted before this time are usually considered to be in the past. The verification of protocols should be careful to prevent any past messages from being being treated as the present. The beliefs are usually assumed to be steady during the whole protocol run.

An encrypted message is denoted as a logical formula bound together and encrypted with an encryption key. The message is transmitted from the sender to the recipient and can be described by using the formula with dynamic property. It is reasonable to halt the verification if we cannot prove the goal is true according to the current premises and assumptions.

The description of several inference rules are as follows.

- The **revelation rule** presents that the message transmitted in plaintext cannot be protected. As to message m, we postulate:

$$\frac{Know(X,\ m), Send(X,\ Y,\ m)}{Know(Z,\ m)}$$

- The **union rule** describes a situation where the conclusion Q of the former action α is the premise of the later action β.

$$\frac{P \vdash_{\alpha} Q, Q \vdash_{\beta} R}{P \vdash_{\alpha o \beta} R}$$

  There is some flexibility to adjust this rule. For example, if a new formula $R \vdash_{\gamma} W$ is inserted behind $Q \vdash_{\beta} R$, a new conclusion $P \vdash_{\alpha o \beta o \gamma} W$ can be obtained.
- The **generation rule** says that if the message m is generated by X, then X must know m.

$$\vdash_{Generate(X,\, m)} Know(X,\, m)$$

The above three rules concern the generation and delivery of messages. In reality, the messages have to be transmitted between the initiator and the responder by encryption. However, we have no idea whether the messages are sent by plaintext or ciphertext. The remaining inference rules are extended from the above rules and the details can be found in the next chapter.

On the other hand, we need to describe how to allocate keys and encrypt and decrypt messages by using symmetric keys and public keys.

- **Encryption by symmetric keys**. We postulate:

$$\frac{Know(X,\, m), Know(X,\, k)}{Know(X,\, E(m,k))}$$

  where k and m represent the symmetric key and message m known by the principal X.
- **Encryption by public key**. We postulate:

$$\frac{Know(X,\, m), Know(X,\, K)}{Know(X,\, S(m,K))}$$

  where K means public keys, which may be a public key-exchange key or a public signature-key.
  In the opposite way, we can have the decryptions by symmetric keys and public keys, respectively.
- **Decryption by symmetric key**. We postulate

$$\frac{Know(X,\, E(m,K)), Know(X,\, k)}{Know(X,\, m)}$$

  This rule says that if the encrypted message m by symmetric key k is known by X and X knows the symmetric key k, then X must know m.
- **Decryption by private key**. We postulate

$$\frac{Know(X,\, S(m,K)), Know(X,\, K^{-1})}{Know(X,\, m)}$$

  This rule says that if the encrypted message m by public key K is known by X and X knows the private key K^{-1}, then X must know m.

The above inference rules are an aid to reasoning about the belief in a security protocol. They are applied in the initial assumptions to conduct a proof or to answer questions about a protocol.

2.2 Security Protocols

In general, the business messages in e-commerce must be electronically transmitted in some manner, and therefore, security services are required to ensure reliable, trustworthy electronic transmission of the messages. As mentioned above (section 1.4.3), the security mechanisms are usually classified into three categories: The above security measures are usually achieved using cryptography and integrated into **security protocols**, which are agreements upon methods of communicating and transmitting data between telecommunication devices. A number of security protocols, including *communication protocols* and *secure transaction protocols*, have been developed to achieve the security objectives of different layers. Figure 2.1 depicts the intermediate role of security protocols. Several security protocols are listed below.

Fig. 2.1. Function of security protocols

- *Transmission Control Protocol/Internet Protocol (TCP/IP)* [64] is a widely used protocol on the internet. TCP operates at the transport layer of the Open System Interconnection (OSI) model, while IP operates at the network layer. The transport layer provides data reliability and integrity checks of the data received. The network layer performs data routing and delivery.
- The protocol that underlies the world wide web (WWW) is called the HyperText Transfer/Transport Protocol (HTTP) [151], which operates on the top of the TCP protocol. Its primary purpose is to define message formats, message transmissions, and web server and browser commands.
- In order to deal with security concerns, the **SSL** (Secure Socket Layer) [54] is responsible for routing messages across networks from their source to their destination. **SSL** adds security, inserting itself between the HTTP application and TCP.
- *Secure Electronic Transaction* (**SET**) *protocol* was developed with the goal of providing a secure payment environment for the transmission of credit card data.

Although most of these are not viewed as secure transaction protocols, they actually contribute to ensure the security of transactions. This book focuses

on the formal analysis of electronic transaction protocols such as SET. We also present the current methods for formal analysis of authentication protocols and cryptography. This provides a comprehensive explanation of the formal analysis of security protocols.

2.2.1 SET Protocol

Secure Electronic Transactions (SET) [140] was jointly developed by the credit-card companies Visa and MasterCard, in conjunction with leading companies in the computer industry such as Microsoft and IBM. It has shown the potential to become a dominant force in assuring secure electronic transactions. SET provides an open standard not only for protecting the privacy but also for ensuring the authenticity, of electronic transactions. It is critical to guarantee success in electronic commerce over the internet. Without confidentiality, consumer secrets cannot be assured, without integrity, the messages between initiator and recipient can be altered, and without authentication, neither the merchant nor the consumer can trust that valid transactions are being made.

Secure Electronic Transactions relies on the techniques of cryptography to preserve the secrecy of messages. There are two different encryption mechanisms used in SET protocol, as well as an authentication mechanism. Symmetric encryption and public-key encryption are used to generate and transmit session keys, respectively. SET simply uses session keys (56 bits) rather than public-key cryptography that can afford the security and protection. The remainder of the transaction depends on symmetric encryption in the form of DES. The public key cryptography is only applied to encrypt session keys and for authentication. The computational cost of asymmetric encryption may be the main concern for substituting it with weak 56 bit DES encryption.

If the session key is used to encrypt a message, then it will be unreadable by unauthorized parties. If the same key is applied to the encrypted message, then the plaintext of the message will be restored. However, we must find a secure way of transmitting the key to all parties. Public-key encryption algorithms use two keys: a public and a private key. Theoretically, it is difficult to deduce the private key from a public one. When we sign a message using someone's public key, only the holder of the private key can read it. Usually, the public key is known by the public. Consequently, this ensures that only the private key holder can read messages encrypted by his/her private key. In the SET protocol, two different encryption algorithms, DES and RSA, are used for the purposes of symmetric encryption and asymmetric encryption, respectively. However, a 56-bit key using DES has been criticized because it can be easily cracked using modern computers. In 1993, a brute-force DES cracking machine was designed by Michael Wiener. As the power of computers

grows, the weak 56 bits DES encryption will become a fatal issue to the safety of transactions.

Authentication plays an important role in SET to ensure that consumers and merchants have faith in the authenticity of each other's messages. Without authentication, any intruder could pose as a merchant or a customer, accept a transaction from a party, and then repudiate any obligation. Authentication is critical to achieve trust in electronic commerce.

SET uses digital signatures to achieve authentication. Using a hashing algorithm, SET can generate a small message digest of the transaction message. It is then signed using the sender's private key. By comparing the transaction message and the message digest, along with the sender's public key, the authenticity of the transaction can be verified. Digital signature helps achieve non-repudiation, since the sender cannot later persuade us that the message was not sent using his/her private key. Privacy of transactions, and authentication of all parties, are important for achieving the level of trust in electronic transactions. However, the encryption algorithms and key-sizes used will be robust enough to protect transited messages from malicious attackers. The idea of the secure electronic transactions protocol is a good start for the success of electronic commerce. However, it remains to be seen whether the protocol is reliable using formal analysis.

2.2.2 Netbill Protocol

Netbill [38] is a protocol designed for micropayment systems for the selling and delivery of information and goods through the internet. It was developed by Carnegie Mellon University in conjunction with Visa and Mellon Bank to deal with micropayments of the online order. As the use of the internet to conduct commerce increases, the internet also poses special challenge due to the lack of standard security mechanisms. However, the convenience makes it a promising and attractive means for future commerce. At the same time, the internet changes the form of traditional transactions. The information transmitted via open networks and distributed systems is subject to observation by malicious attackers, and the transaction data recorded by a merchant's computer gives rise to privacy problems. On the other hand, parties may want to authenticate each other, negotiate the price, choose the means of payment, or control the access.

The NetBill transaction model includes three parties: the customer, the merchant and the NetBill transaction server. The NetBill server acts as both a certification authority and a payment intermediary linked to conventional financial institutions. In addition, it also takes charge of the distribution of public/private key pairs as well as the symmetric keys that are used to encrypt the exchange between the customer and the merchant, the customer and the server, and the merchant and the server.

A Netbill transaction consists of three phases; price negotiation, goods delivery, and payment. In a transaction, the customer and merchant interact with each other in the first three phases; the NetBill server is not contacted until the payment phase, when the merchant eventually submits a transaction request. Each phase has a collection of transaction objectives so as to provide authentication, privacy, payment, delivery or access control.

Before starting the transaction, the customer and merchant need to register their public/private key pairs and user identifications with the Netbill server. Before establishing communication channels, a mutual recognition occurs to allow the parties to authenticate each other in terms of the public key Kerberos system. A session key is established between the customer and merchant by virtue of a session ticket and a certificate.

- **Negotiation** is the phase where the customer requests a price quote from the merchant, and the merchant responds with a specialized quotation according to the customer profile. Identification and authentication of the customer is required in this step.
- **Goods delivery** is applied once the customer agrees with the negotiated price with the merchant. A new symmetric encryption key is generated to encrypt the goods without including the key in the message. The decryption key is delivered to the customer only when the merchant confirms the received payment.
- **Payment** ends the transaction by depositing the payment with the merchant and sending the decryption key to the customer. In this phase, an Electronic Payment Order (EPO) is applied. It includes transaction information and payment instructions, the former is readable by the merchant and Netbill server, and the latter is only known by the Netbill server.

In contrast with other electronic transaction protocols, Netbill has some particular properties. The customer is only charged after receiving the encrypted information, and the corresponding decryption key is only delivered after payment.

2.2.3 Security Services

Here we give a description of the general security services that security protocols are usually required to provide. Although many security terms have been widely used in the literature, we can see various interpretations of them, which may cause confusion. A coherent definition may avoid the vagueness and be of benefit to the understanding of the security properties of protocols. As mentioned above, the security properties largely depend on cryptography. It is not easy for us to find a security protocol that does not include any cryptographic techniques. Nevertheless, as the computation power of modern computers advanced, so did the complexity of cryptography such as base-64

encryption versus base-128 encryption. A protocol can only be claimed to be secure if it can satisfy the required secure properties under the assumption that the applied cryptographic algorithms are robust and trusted.

In the open network and distributed systems, messages must be transited in some secure manner. Several important security services are required to ensure reliable, trustworthy electronic transmission of messages. The primary security services can be roughly divided into five categories:

- Confidentiality: When a message is sent electronically, the sender and receiver may desire that the message remain *confidential* and thus not be read by any other parties. In other words, the information is accessible only to those authorized to have access. Confidentiality is one of the design goals of many cryptosystems, and is realized in practice via modern cryptography. In a more strict interpretation, the high-level users should not know the secrecy of low-level users. It should be immune to any performed traffic analysis. For example, in the Needham Schroeder protocol, though the trusted server S may know *Alice* wants to talk to *Bob* and maybe even knows *Bob* agrees with *Alice*'s request, the server is unable to read the contents and trace the communication between *Alice* and *Bob*. This property states which behaviours are permitted and which are not. It must be formulated as a set of system behaviours rather than an individual predicate during formalization.

  We can illustrate the confidentiality using an example in e-commerce. Suppose X and Y are principals who want to talk with each other, and ST denotes a set of sensitive messages such as order details and credit card information, that should be kept confidential under the protection of cryptography. Let K represent a session key between X and Y, which is issued by a server S. What the confidentiality means here is that the form of plaintext of messages in ST should not be known by an intruder. Although the intruder perhaps obtains the sensitive messages under encryption, he/she is unable to decrypt them without the corresponding key. However, the protection over confidentiality can be breached, probably X or Y divulges the key, as long as the intruder gets hold of the decryption key.
- Integrity: When a message is sent electronically, the integrity denotes that the message is not modified or corrupted during its transmission. A message that has not been altered in any way, either intentionally or unintentionally, is said to have maintained its integrity. The integrity can be compromised in two primary ways. Malicious alteration denotes the intentional corruption of messages, such as altering personal bank account details maliciously, and forging a credit card transaction. Accidental alteration denotes the unintentional corruption of messages, such as communication block and hardware outrage. Considering data integrity, one important aspect is to assure that the data can be accessed and modified

by those who are authorized to do so. If a modified message is accepted by the recipient, it indicates the violation of this property. In reality, the integrity checking usually happens along with the entity authentication and the other alike authentications, but they play different roles. The former guarantees the validity of messages but the authentications assure that the other party is alive and whom he claims to be.

In an electronic transaction, the payment information sent from purchasers to vendor includes order information, personal data, and payment instructions. One of the major security concerns is to ensure that it is not alerted. If any component is modified during transit, the transaction will not be processed correctly. We must provide the means whereby the contents of payment information received match exactly the contents of the message sent. An effective method to assure this security property is by the use of digital signature based on an one-way cryptographic function, namely hashing. To generate the digital signature of a message, we must obtain the digest of a message using the hashing function. The message digest is encrypted using the sender's private key and is appended to the original message. The result is known as the digital signature of the message.

- Non-repudiation: Non-repudiation denotes the assurance that a transferred message has been sent and received by the principals claiming to have sent and received the message. It is a means to guarantee that the sender of a message cannot deny having sent the message and that the recipient cannot deny having received the message. Well-designed electronic transaction systems provide non-repudiation, which is the provision of irrefutable proof of the origin, receipt, and contents of an electronic message. The most effective way to enable it is through the combined use of hashing in both transactional directions and digital signing. In addition, transaction certificates, timestamps that contain the date and time a document was composed, and prove that a document is valid at a certain time, and confirmation services to indicate that messages were sent and/or received, all help to provide non-repudiation.

  To achieve non-repudiation, we must confirm that one participant is protected against possible fraud by another. This requires that both participants keep evidence with respect the transactions occurring . Moreover, it is insufficient for any participant to provide the required proof. The proof must be able to convince a trusted third party. With the collected evidence, if we can indicate that the recipient could have received a message in case the sender had sent the message to him/her, then we can say the non-repudiation property holds. As already described, the digital signature mechanism can be used to sign a message from *Alice*, so *Bob* can confirm that the received message is indeed sent from *Alice*. A trusted

third party is an alternative way to achieve this goal, but this may impose extra communication costs.

- Data Authentication: This security service not only assures that the message to a receiver is indeed originated by the correct sender, but also confirms that message has not been altered. In some cases, the freshness is also required for data authentication.

  In a protocol run, one process may be the authentication where *Bob* authenticates that the message was really sent by *Alice* and is still in the period of validity. On the other hand, *Bob* needs to assure that the message has not been tampered with during its transit. These two aspects are two primary components to guarantee data integrity. In reality, the authentication may involve several authentications back and forth such as the one-way authentication, two-way authentication and three-way authentication in [23].

  Although we have not talked much about the freshness of messages, this is actually an important issue in data authentication. *Bob* may not believe in an outdated message from the sender, but wants the message from *Alice* to have been sent recently. Timestamp is an efficient way to warrant the freshness. To obtain more detailed information about the application of timestamp, we can refer to Denning's model in [40].
- User Authentication: The most widely used form of user authentication is by passwords in conjunction with identification code (ID). In practice, the verification of the ID can use physical means such as identity card of the user, or calling a trusted authority or using the user's PIN. The user authentication is usually combined with data authentication. Thus we are able to verify that the message is originated from the purported principal and that the message has not been altered. If the identity authentication is passed, a secure communication channel is established between the principals.

  When an electronic message is received by a user or a system, the identity of the sender needs to be verified in order to determine if the sender is who he claims to be. In some case, trusted third-party services are engaged to 'vouch for', or authenticate the user. Common authentication measures are digital signatures, challenge-response, password, smart card, and tokens. A notion of timeliness that is ignored in the previous formalization needs to be considered. It represents that the user's identity authentication is a one-off process. In other words, it is unallowable to go on to another new transaction under the assumption that the previous identity authentication is still valid and makes sense.

  It is actually not easy to include the user authentication into the formal formulae. In practical formal analysis, this authentication is usually integrated into the data authentication at each step of the interaction between principals.

- Key Authentication: When using public-key cryptography, the key authentication can arise. In traditional symmetric cryptography, the encryption key is trivially related to the decryption key, in that they may be identical or there is a simple transform to go between the two keys. The keys, in fact, denote a shared secret between two or more parties that can be used to maintain a private communication link. Key authentication was not an issue in symmetric cryptography because it uses some methods of key distribution to assure authenticity. However, the public-key cryptography cannot avoid this issue. The public key can be known by any of the principals without compromising the encryption algorithms, which can result in some attacks by forging a reagent public key. For example, an intruder may claim that a key is *Alice*'s public key, but it is actually known by the intruder. To enable authenticated, confidential communication across open networks, we need to ensure both the secrecy and authenticity of the public-key. In other words, the public key is only known by the authorized principals, and is verified to be valid.

  An alternative to this issue is the use of certificate authorities (CA) via a public key infrastructure (PKI) tree to authenticate that the public key belongs to the claimed principal. The Certificate Authority authenticates a principal's claims according to its specified policies. For example, this Certificate Authority may require *Alice* to present a driver's licence or passport for an online transaction before it will issue a certificate. Once *Alice* has provided proof of her identity, the Certificate Authority creates a certificate containing *Alice*'s name and her public key and digitally signs it. It contains not only owner identification information, but also a copy of the owner's public key. The public key of the Certificate Authority may be known to as many people as possible. Thus, by trusting a third party and a single key, a key authentication can be achieved in which one can have a high degree of trust. Meanwhile, this method relies on a certificate issued by Certificate Authority. Thus, any defective certificate can give rise to some problems with respect to the key authentication.

  Key authentication has been widely used in many security protocols, like the recent SET protocol [138, 139, 140] to secure payment card transactions over open networks. A representative example of this is a portion of the purchase request, in which the authentication takes place through the merchant M transferring an initiate response to the cardholder C according to the following steps:
  - M generates an initiate response *IniRes*.
  - M digitally signs it by generating a message digest of the *IniRes*, and encrypts it using the merchant's private signature key, namely $\{Sign(H(IniRes),\ Spv(M))\}$, where $H(IniRes)$ denotes the one-way hashing function, and $Spv(M)$ denotes the merchant's private signature key.

- M sends a response along with the merchant and payment gateway's certificates to the cardholder C: $m = \{Sign(H(IniRes),\ Spv(M)),\ Cert(M),\ Cert(P)\}$, where P, Spv and $Cert$ represent the payment gateway, private signature key and certificates, respectively.

M sends to C the message:

$$\{Sign(H(IniRes),\ Spv(M)),\ Cert(M),\ Cert(P)\}$$

After receiving this message, the cardholder carries out the following operations:

- C obtains the public keys of M and P, namely $Spb(M)$ and $Spb(P)$ from the certification path of PKI, after verifying that the certificates of M and P did not expire by traversing the trust chain to the root key, where Spb denotes public signature key.
- C verifies the merchant's signature by decrypting the signature with the merchant public signature key $Spb(M)$. C then verifies that the obtained result is identical to the newly generated message digest of the initiate response $IniRes$. Thereby, it is able to ascertain the validity of the signature and the integrity of the signed message.

These exchanges assure the following:

- The authenticity of M and that the public key $Spb(M)$ was authorized by CA.
- The authenticity of P and that the public key $Spb(P)$ was authorized by CA.
- The integrity of the initiate response message $IniRes$ by M.

In reality, we may need to verify additional security properties using similar exchanges but in different authentication ways and directions. For example, the Needham and Schroeder protocol and Yahalom protocol, both key distribution protocols, generate the key via a server. However, the Diffie-Hellman protocol is a key agreement protocol, in which the participants all contribute to the resulting keys.

- Authorization (also known as access control): Computer systems require a certain amount of data sharing. Limiting access to data and systems only to authorized users is the main objective of authorization. The certificate, 'privilege management', and firewall technology are effective forms of authorization. This process helps to identify principals. When a principal intends to access specific information or resources, the authorization process validates that the principal has been granted permission to use that information or resources. Authorization may be based on various restrictions such as time restrictions, credit limit or restrictions where 'Anonymous consumers' or 'guests' have very few permissions.
- Freshness: Freshness guarantees against replication of messages or keys, where a message replay attack may happen. During the message exchange between principals, an old message or key can be replayed by an intruder

if the message freshness or key freshness is not assured. In Message 4 of the Needham-Schroeder protocol, B provides key confirmation to A, since the inclusion of n_B means that the message originates from B, and hence B must know and agree with the key K_{AB}. However it should be clear that this protocol does not provide key freshness, because B has not told A whether this message has just been generated by B, or is just a replay message generated before by B. The lack of freshness can usually be ameliorated using timestamp or a random sequence number within the scope of the encrypted message.

- Fairness: With the growth of open networks, many electronic transaction protocols have been proposed and applied, in which the requirement of fair exchange of electronic items has become prevalent. In this type of protocol, no party should be able to have an advantage over another party when exchanging messages. Two or more parties can exchange their information in a fair way. Fairness is not required for non-repudiation, but it probably would be desirable in some cases under the requirements of the participants. For example, in the purchase request process described above, the cardholder could refuse to continue the payment after the merchant has signed and agreed with the purchase request from him/her. This security service has been discussed and defined in recent works such as [107, 109]. Trusted third parties are a useful tool to address this issue, but this can result in an increase of computations and/or communications.
- Liveness: A security protocol may either rely on or try to assure liveness or fairness properties as well as safety properties. A liveness property states that something good will finally happen in contrast to a safety property where something bad will never happen. This property guarantees that each request from any party is followed by the corresponding service, and each run of protocol is eventually terminated. To evaluate the liveness property, it is necessary to keep track of the visited states of the protocol, and the order of these states. For example, it is probably necessary for a transaction system to receive an acknowledgement for each message that it sends. In [77], the liveness property is expressed in temporal logic, and a model checker was developed to analyse a security protocol for assuring fair exchange of digital signatures.
- Non-repudiation: This security property guarantees that a transaction cannot be later denied by one of the parties involved participants. In electronic transaction systems based on open networks, this property plays an important role because the agreement on a transaction has to be completed through networks rather than in the traditional face-to-face manner. It intends to provide all parties with evidence that a certain run of protocol has really occurred. It is able to prove that a message has been sent by the sender and it has been received by the claimed responder. For example, the merchant M sent a signed initiate response to the cardholder C in response to C's purchase request. It is essential that we have sufficient ev-

idence to prove that the message was sent by M and received by C shoud either of them attempt to deny it.

Clearly, traditional methods such as seals or signatures are easy to forge. The non-repudiation of digital information can be assured using signature-based mechanisms. Furthermore, the development of biometric techniques, such as retinal scanning and finger printing, and DNA encryption, will perhaps provide non-repudiation.

- Anonymity: Anonymity is a security property that an individual's true identity is unknown in a certain situation. Suppose there are a set of arbitrary elements including principals and messages, if an element is unidentifiable, then the element is said to be anonymous. For example, A wants to communicate with B but he/she does not want B to know his/her real identity. The degree of anonymity can be varied under different scenarios. For example, if a message is only known by A, B and C, each of them has equal opportunity of revealing this message, but we cannot determine whether it is A, B or C who did it. Nevertheless, if A has plausible evidence to prove he/she is innocent, then we may deduce that it must have been B or C who disclosed the message. Although the anonymity still remains, it is easier to decide that he/she is either B or C.

  If a set of messages are shared by a number of parties, the degree of anonymity is higher with respect to these messages. However, this may result in a serious attack, namely collusion attack, in which a principal can perform an attack in conjunction with a certain number of dishonest principals (see Chapter 6).

  Many examples of systems showing different degrees of anonymity have been analysed using FDR, and the CSP, a possibilistic method, was also used to formalize this property, in which the renaming operator of CSP is applied to shuffle a set of events E. If the observed system is unchanged after an arbitrary shuffling of the set E, then the system is able to provide anonymity.
- Availability: Regardless of the above security properties to protect messages in transit from malicious attacks, a system is also needed to provide correctly and promptly, the required services to authorized users. Whatever key-establishment protocol or a key-distribution protocol we formalize, we would like to ensure that a public key pair or a session key is really established between communicating parties. As a result, if A wants to talk with B through a trusted server S, S must be able to work promptly and subsequently generate a session key for A and B and assure that A and B both know this fresh session key.

  Certainly, availability is just used to serve the legal users rather than for malicious intruders. Thus, we must prevent intruders from utilizing the available services to achieve their illegal purposes. The formalization of a security protocol needs to take this into account.

2.2.4 Principles of Cryptography

In general, the messages in electronic transactions must be electronically tranmitted in some manner, and therefore, security services are required to ensure reliable, trustworthy electronic transmission of the messages. The security mechanisms are roughly classified into three categories [48, 59, 102]:

- System security means nothing happens to the computer and equipment, including virus, logic bomb etc.
- Network security means only the authorized users can gain access to the network and do what they can do based on the privilege assigned to them.
- Data security means how to ensure the integrity and confidentiality of transited messages over open networks.

The primary method used to achieve data security is encryption [13, 23], which is a process of encoding a message so that the meaning of the message is not obvious. The reverse process is called decryption, transforming an encrypted message back into its normal form.

The original form of a message is usually known as **plaintext**, and the encrypted form is called **ciphertext** [130]. The set of all the plaintext messages is denoted by M_P; similarly the set of all the ciphertext is denoted by M_C, and f is a mapping from the variables in M_P into the set M_C. P and C represent **plaintext** and **ciphertext**, respectively. Both of them are stored and transited in binary data. They can be expressed in mathematical formulae:

$$C = f(P) \tag{2.1}$$

In the reverse process, the decryption function f^{-1} operates on C to obtain plaintext P:

$$P = f^{-1}(C) \tag{2.2}$$

where $P \in M_P$, $C \in M_C$.

Actually, f is viewed as the encryption algorithm (function) and f^{-1} denotes the decryption algorithm. Since the encryption and decryption are inverse functions of each other, the following formula must be true:

$$P = f^{-1}(f(P)) \tag{2.3}$$

There are two important classes of encryption algorithms - *symmetric cipher* and *asymmetric cipher* - which aim to achieve the goal of data security as follows.

Symmetric cryptography (secret-key cryptography or single-key cryptography) where entities share a common secret key is depicted in Figure 2.2. This key must be kept secret because anybody who obtains this key can encrypt or decrypt messages unrestrictedly using this key. In symmetric key

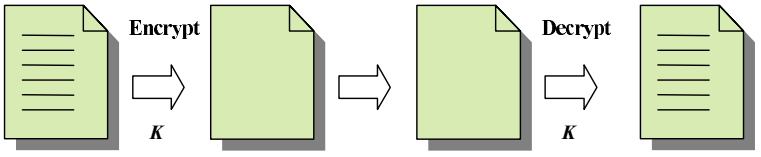

Fig. 2.2. Symmetric cryptography

systems, the representative encryption algorithm in use is the Data Encryption Standard (DES) [153]. Usually, the encryption and decryption processes with a symmetric key are represented by:

$$C = f_k(P) \tag{2.4}$$

$$P = f_k^{-1}(C) \tag{2.5}$$

Symmetric cipher can be classified into two categories, block cipher and stream cipher. The former operates on a certain number of bits, such as 64 bit or 128 bit blocks in the DES cipher, and the latter operates on a bit at a time, such as the classical Caesar cipher. Both ciphers use the same key for encryption and decryption. In a stream cipher, only one bit is influenced by the bit error during transmission. In contrast, in a block cipher, the whole block will be affected in case of one bit error. Although stream ciphers can provide higher security than block ciphers they have been criticized for the weakness in assuring integrity. Since a stream cipher encrypts each bit of the plaintext at a time it may result in the alteration of the ciphertext, such as inserting or deleting letters, in a manner that is not easily detected by users. For example, in a financial transaction, a customer wants to pay $100 for an online order. The amount of money could be changed to $150 but it is not readily detectable. Despite the difficulty of doing that, it still indicates a potential danger of the increasingly rapid growth in computing power.

Classical ciphers consist of substitution ciphers and transposition ciphers. A good cryptography system actually uses both of them.

A *substitution cipher* is the cipher in which each bit in plaintext is substituted for another bit in the ciphertext in terms of a known principle of substitution, by which the recipient is able to invert the substitution and recover the plaintext. The different types of substitution ciphers are introduced in [23]. Caesar cipher is one of the representative ciphers. It is based on a code book and replaces each plaintext character by a specified character. For example, C is replaced by H, F is replaced by J, and Z is replaced by D, according to the principle in which each character is replaced by the character four to the right *mod* 26.

A *transposition cipher* is one in which the characters in the plaintext are reordered rather than substituted for new characters. Usually, the transposi-

tion is operated within fixed length blocks of the plaintext. For example, let $P =$ 'A E J L E A R T H' be the plaintext. Suppose the block size is equal to 3, and the permutation is (3 1 2), which denotes that the first character is rearranged to the third position of the ciphertext, the second character is rearranged to the first position of the ciphertext, and the third character is rearranged to the second position of the ciphertext, respectively. Eventually, the ciphertext is $C =$ 'E J A L E A R T H'.

Regardless of its wide application in cryptosystems, this cipher involves a considerable consumption of memory and requires every message to be a multiple of a specified length. On the other hand, the frequency of the characters in ciphertext can be utilized by a cryptanalyst to estimate the plaintext because the frequency of the characters in ciphertext is identical to that in the plaintext.

Public key cryptography, namely asymmetric cryptography, where each communicating entity has a unique key pair, a public key and a private key, is depicted in Figure 2.3. It was introduced by Whitfield Diffie and Martin Hellman [42]. Diffie-Hellman cryptography is still used today, but it has some vulnerability to a man-in-the-middle attack. In asymmetric cryptography, the key used for encryption is different from the key used for decryption. Thus, it is infeasible to derive the decryption key from the encryption key due to the computational complexity. The public key (encryption key) is often made public, and the private key (decryption key) is only known by a certain party and must be kept private. The recent SET protocol proposes a public key-exchange key and private key-exchange key, in which messages will be encrypted with the private key and decrypted with the public key. This is often used to generate a digital signature. The RSA algorithm by Rivest, Shamir and Adleman is a prevalent scheme for digital signature. To avoid possible confusion, k and k^{-1} are used to denote the encryption key and decryption key rather than the public key and private key, respectively.

$$C = f_k(P) \tag{2.6}$$

$$P = f_{k^{-1}}^{-1}(C) \tag{2.7}$$

Although asymmetric cryptography is important to achieve user authentication and key management, it is actually not an efficient way to exchange messages owing to the considerable efforts required for computations. On the contrary, symmetric key algorithms are generally much less computationally intensive than asymmetric key algorithms. The disadvantage of symmetric key algorithms is the requirement of maintaining one copy of a shared secret key at each end. As keys are vulnerable to being discovered by a malicious cryptanalyst, they need to be altered frequently and kept private whenever distribution and in use. Consequently, it is hard to select, deliver and store keys without errors and loss.

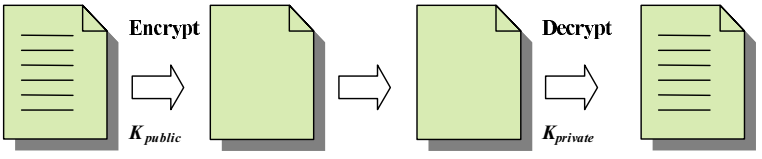

Fig. 2.3. Public-Key cryptography

As a result, most cryptosystems use the much slower asymmetric algorithms to distribute symmetric-keys at the beginning of a session, then the higher speed symmetric-key algorithms take over. Regardless of the reliability issue in asymmetric key distribution, they tend to be more tractable.

The issuing of a **certificate** is used by one party to obtain the public keys of other parties and assure that they are valid and associated with the right parties. This is usually realized by establishing Certificate Authority (*CA*), which is viewed as a trusted third party and takes charge of authenticating public keys and associating them to the correct users. Certainly, *CA* must be trusted by all parties.

A certificate may consist of the user's personal details, such as name or ID code, the user's public key and lifetime, by which we can associate the user's public key to the correct user. Usually, the recipient needs to authenticate the received certificate from the sender, which is linked to the entity that signed it. By following the trust tree of public key infrastructure (PKI) to a known trusted party, one can be assured that the certificate is valid. Figure 3.2 in Chapter 3 illustrates the hierarchy of trust. The public key of the root is known to all parties and may be used to verify each of the certificates in turn.

The problem of assuring a correct match between entity and public key is difficult in practice. Thus additional authentication techniques need to be integrated together with *CA*, such as external reliable third parties. Furthermore notaries may be required in some cases to authenticate personally the party whose signature is being verified. Another problem that may be confronted with this technique is due to the objective difficulties of setting up the lifetime for certificates. It is not easy to give an appropriate value and may result in negative influence in case it is too long or too short.

One-way hash functions are the functions that are easy to compute but are extremely difficult to reverse. A hash function takes a string of any length as input and converts it into outputs of some fixed length. Sometimes the output is called as 'message digest' or 'digital fingerprint'. One of its properties is that a small change on the input can result in a significant change in the hash value obtained. It is widely used to assure authentication and integrity. Let H be a hash function and M be a message. The hashing process is described below:

$$H(M) \tag{2.8}$$

MD5, SHA1 and RIPEMD-160 are three of the commonly used hash functions. However, they have been reported to have some flaws in recent studies. A hash function must meet two requirements if it is considered to be secure. Given a message digest, it should be infeasible for an intruder to find a message that matches the given digest. Also, it should not be possible to find two messages whose digests are very similar or equivalent.

A typical use of a cryptographic hash would be digital signature. For security and efficiency reasons, many digital signature algorithms specify that only the digest of the message is 'signed', but not the entire message. Hash functions convert the plaintext of arbitrary length to a specified length for a block cipher. Also, the converted messages can be sent to the recipient without revealing it. Furthermore, hash functions can be served as a way to store some sensitive text or value, such as password and PIN.

Digital signature is designed to bind the message originator with the exact contents of the message. The sender uses his/her private key to compute the digital signature. In order to calculate the digital signature, a one-way hashing algorithm (such as SHA-1 [134]) may be used initially to calculate a message digest. The sender's private key is used at this point to encrypt the message digest. The encrypted message digest is what is commonly referred to as a digital signature.

RSA is one of the well-known algorithms to implement digital signature in conjunction with hash functions. As we described above, the latter is applied to transfer the plaintext to the block of certain length specified by the encryption algorithm, and the former is used to encrypt the converted text. Suppose X and $Spv(X)$ denote the sender and his/her private key, respectively. Let Y be the recipient. Then the encryption is as follows:

$$Sign(H(M), Spv(X)) \tag{2.9}$$

The message M can be X's telephone number, bank account or similar things. To help the recipient confirm the message is authentic, X's identity and M are usually appended to the original message to construct a digital signature.

$$X, M, Sign(H(M), Spv(X)) \tag{2.10}$$

When the recipient Y receives this message from the sender X, Y needs to confirm that it has not been altered and was indeed signed by the X's private key. The authentication consists of three steps:

- Generate the hash of message M, namely $H(M)$, using the publicly known hash function.
- Apply X's public key to decrypt the encrypted hash of M in terms of the described public-key cryptography: $\{Sign(H(M),\ Spv(X)),\ Spb(X)\} \Rightarrow H(M)$

- Compare the generated hash in the first step and the obtained hash in the second step.

If the message M was not encrypted by the correct private key of X, or two hash values are not identical, the authentication will fail. Therefore, it is essential to prevent the forgery or compromise of X's private key and the hash of messages. It should be extremely difficult for an intruder to obtain the private key and generate an equivalent hash using a given message. The decryption of messages encrypted by X's private key should be computationally infeasible. Also, changing even one letter in the raw message changes the message digest in an unpredictable way.

Although some approaches have been used to guarantee the private key is really associated with X, there remains the problem that X can reveal his/her private key to an intruder. In that case, we cannot assure the message was indeed signed by X rather than an intruder. This issue is hidden but is highly dangerous. It is called collusion attack in this book and will be discussed in the course of the following chapters.

Timestamps are one of the important security services to guarantee the freshness of delivered text or value. For some specific applications, such as electronic transactions, people may want to add a timestamp along with the delivered message to certify that the message is valid on a certain time. Then the receiver can assure this message is still fresh within a certain time by checking the attached timestamp against the local time. Certainly, it is important to maintain the consistency between clocks across the whole network. To assure the accuracy of timestamps, we often derive the time from reliable servers.

If we use timestamps in security protocols, it is necessary to be wary of the forgery of timestamps by malicious intruders. One way to solve this problem is to generate hashed timestamps and link the current timestamp with the previously received timestamps. This is effective not only to prevent the intruder from forging an equivalent timestamp but also to prevent the collusion of producing timestamps between the sender and receiver.

Despite the difficulties of obtaining accurate timestamps and ensuring the validity of the timestamps in distributed systems and open networks, it is still an effective way to assist in authenticating messages that is valid within a certain time. The Time Stamping Protocol (RFC3161) is a protocol that defines the entities involved (the requestor and the Time Stamp Authority or server), the message format and the communication between the entities.

Apart from the above methods, the **access control**, such as *password*, *fingerprint* [161], and *smartcard* [80] can assist in identifying authorized users and providing authentication. In addition, another powerful method is *firewall software* [104]. Erecting a firewall between sensitive transaction information and untrustworthy networks is essential to prevent security break-ins through vulnerabilities in the operating system.

These security services have been deployed to relieve the security concerns. Among them, data and transaction security are critical. Without them, information transmitted over the internet is susceptible to fraud and other misuse. Hence the security of electronic transactions must be recognized and dealt with when performing transactions over the open network.

2.2.5 Threats in Security Protocols

To enumerate completely all threats to which security protocols are subject may be difficult and impractical. Although some traditional security issues are important such as security management and policy, risk management and security and law, they are not considered in this book. We mainly focus on discussing the security threats that may have an influence on the formal analysis of security protocols. Furthermore, we need to pay attention to the emerging issues in security protocols due to varied threats in complex protocols, such as denial of services and traffic analysis, hence we not only deal with the detection of security vulnerabilities in protocol design.

Man-in-the-middle attack (or **MITM**). This attack utilizes the vulnerabilities of security protocols to masquerade as one of the two parties to the communication. For example, in public-key encryption, suppose A wants to communicate with B, and that C wishes to deliver a false message to B during the conversation. The protocol starts with the request of B's public key by A. If B sends his/her public key to A, but C is able to intercept it, a man-in-the-middle attack can begin. C can simply send A a public key for which he/she has the valid private key. A, believing this public key to be B's, then encrypts his/her message with C's key and sends the encrypted message back to B. C again intercepts, decrypts the message, and encrypts it (after modification if desired) using B's public key as originally sent to A. B will believe this message is really sent to him/her by A.

The original Diffie-Hellman key exchange protocol has been found to be vulnerable to the man-in-the-middle attack, when used without authentication.

Suppose *Alice* and *Bob* want to generate a shared secret key but they do not know each other's private key. To start the protocol, *Alice* and *Bob* need to agree to use a prime number p=23 and base g=5. A brief example of this protocol can be

- *Alice* selects a secret integer a, then sends *Bob* ($g^a \bmod p$).

$$Alice \rightarrow Bob\text{: } g^a \bmod p$$

- *Bob* selects a secret integer b, then sends *Alice* ($g^b \bmod p$).

$$Bob \rightarrow Alice\text{: } g^b \bmod p$$

- *Alice* calculates $(g^b \bmod p)^a \bmod p$
- *Bob* calculates $(g^b \bmod p)^a \bmod p$

As a result, *Alice* and *Bob* receive the same value, namely $(g^b \bmod p)^a \bmod p = (g^b \bmod p)^a \bmod p$, due to $g^{ab} = g^{ba}$. a, b and $(g^b \bmod p)^a \bmod p$ serve as *Alice*'s private key, *Bob*'s private key and the shared secret key between *Alice* and *Bob*, respectively.

If the values of a, b and p were big enough, it should be difficult for *Alice* to know *Bob*'s private key or for *Bob* to know *Alice*'s private key. However, this key exchange protocol is vulnerable to a man-in-the-middle attack. In this attack, an intruder Z intercepts *Alice*'s public value and sends his/her own public value to *Bob*. When *Bob* transmits his public value, Z replaces it with his/her own and sends it to *Alice*. Z and *Alice* thus agree on one shared key that is different from the key shared between Z and *Bob*. After that, Z is able to decrypt any messages exchanged by *Alice* or *Bob*, and even alter them before re-encrypting with the appropriate key and delivering them to the other party. This vulnerability has arisen because this protocol does not authenticate the participants.

This example indicates the need for us to have some way to ensure that A and B are indeed using each other's correct public keys. Otherwise, such attacks may damage any message sent using public-key technology. Fortunately, a number of techniques have been developed to protect messages from man-in-the-middle attacks, such as password and stronger mutual authentication.

Replay is one kind of attack in which a valid message or data is maliciously repeated or retransmitted.

Suppose *Alice* requires *Bob* to prove his identity using a hashed *password*. A possible replay attack is described below.

- Message 1 $Bob \rightarrow Alice$: $H(password)$
- Message 1′ $Z \rightarrow Alice$: $H(password)$

The intruder Z intercepts the *password* and retransmits it to *Alice* after the exchange is over.

There are two primary ways to prevent replay attacks. One way is to use session tokens - *Bob* can combine a one-time token from *Alice* with the *password*.

- Message 1 $Bob \rightarrow Alice$: $H(token,\ password)$

If two session tokens match, *Bob* is allowed to log in. In this case, the intruder Z cannot replay the session token next time because *Alice* may use a new session key.

Timestamp is another way of avoiding replay attacks. Synchronization is important to keep the consistency of time of different parties. When *Bob* sends a message to *Alice*, he includes the time broadcast by *Alice* in the message m.

The message m is only believed to be valid if the timestamp is within an allowable range of tolerance.

Denial of service (or **DoS**) is an attack on a computer system or network that results in the excessive consumption of computational resources, damage of configuration information, or disruption of connection. For example, the consumption of memory and bandwidth, the malicious alteration of routing information, and the disconnection of access service from users. A denial of service attack does not usually give rise to the theft of information or other security loss, however, a user or organization might not have access to a particular service they would normally expect to have. Sometimes it may cost the user or company considerable time and money.

Interleave is an attack that uses several runs of the protocol to masquerade their interactions. The Needham-Schroder public key protocol is used to provide mutual authentication between A and B by secretly exchanging two nonces:

- Message 1 $A \rightarrow B : \{A, n_A\}_{Spb(B)}$
- Message 2 $B \rightarrow A : \{n_B, n_A\}_{Spb(A)}$
- Message 3 $A \rightarrow B : \{ n_B\}_{Spb(B)}$

At the first step, B is able to confirm the n_A from A, but A has not been convinced of B's identity. B then generates a n_B and sends the encrypted n_B and n_A to A. At the last step, B checks the n_B is correct. This protocol was analysed by BAN logic and has been believed to be secure. However, the following interleave attack remains possible in the protocol. Let Z be an intruder.

- Message $1'$ $A \rightarrow Z : \{A, n_A\}_{Spb(Z)}$
- Message $1''$ $Z \rightarrow B : \{A, n_A\}_{Spb(B)}$
- Message $2'$ $B \rightarrow Z : \{n_B, n_A\}_{Spb(A)}$
- Message $2''$ $Z \rightarrow A : \{n_B, n_A\}_{Spb(A)}$
- Message $3'$ $A \rightarrow Z : \{n_B\}_{Spb(Z)}$
- Message $3''$ $Z \rightarrow B : \{n_B\}_{Spb(B)}$

A starts a protocol run with Z. However, Z uses the n_A to initiate another run with B rather than answering A. B responds with n_B and encrypts it using A's public key. As Z cannot decrypt this encrypted message, he/she just forwards it to A. A decrypts it and responds with encrypted n_B using Z's public key. Thus, Z now gets to know n_B now. Eventually, B also obtains the expected response n_B from Z.

Message $1''$ and Message $2'$, and Message $2''$ and Message $3'$ are two interleaved runs of the protocol conducted by Z. At the end of this authentication, both A and B will think they share the n_B and n_A with Z. This attack is beyond the assumptions of BAN logic. One solution is to link messages in a run with chained nonces.

Reflection is an attack of tricking the participant into answering his/her own questions in a challenge-response protocol. Suppose there is a server S to which users want to log on. The challenge-response is performed to authenticate the users instead of sending a password through the open network. Let Z be an intruder, k be a symmetric key shared by Z and S, and e be a cryptographic function.

- Message 1 $Z \rightarrow S : \{ n_Z\}$
- Message 2 $S \rightarrow Z : \{e(n_Z, k), n_S\}$
- Message 3 $Z \rightarrow S : \{ n_S\}$
- Message 4 $S \rightarrow Z : \{e(n_S, k), n'_S\}$
- Message 5 $Z \rightarrow S : \{e(n_S, k)\}$

In the second message, S responds with his/her nonce n_S. However, Z utilizes n_S to fake a new request. At the fourth step, Z obtains the encrypted n_S and sends it back to S in a later answer. As a result, this leaves the intruder Z a fully-authenticated connection but the other session is simply abandoned.

Solutions would be to require the initiator Z initially to respond to challenges before the server S answers its challenges, or require the key to be different between the two directions, or reject encrypted nonce n_S that was just sent out for client authentication.

Algebraic is a cryptographic attack that intends to break the underlying cryptographic algorithms. Aside from the described logical attacks that utilize the weakness in protocols, we should also be wary of the algebraic attack. Without exception, every cryptographic algorithm has algebraic identities as the consequence of its mathematical infrastructure. Some of them may be already known, or unknown until now. Such identities may provide a good opportunity for attackers to undermine the security of the protocol. Therefore, it is necessary to take these identities into account to enable the modelling framework to represent such identities.

Traffic analysis is the process that focuses on interpreting and inspecting messages so as to deduce the information patterns in communication and determine the source and destination of message traffic. No matter whether or not encryption is used, this can be performed. It can be possible for an observer to infer a great deal from the observed or even intercepted messages.

Traffic analysis is a key issue in computer security. An intruder can obtain important information by monitoring. For example, an attack on the SSH protocol used timing information to deduce information about passwords [141]. The authors use hidden Markov models to study the timings between messages, and attempt to recover the password. Traffic analysis can be also used for attack on anonymous communication systems, like Tor (anonymity network). Murdoch and Danezis presented traffic-analysis techniques that allow opponents with only a partial view of the network to infer which nodes are being used to relay the anonymous streams [118].

Regardless of the number of dedicated software systems that have been developed to deal with this problem, they still lack the ability to measure the degree of protection provided by these programs. It is not easy to evaluate what level of security they are able to provide. Therefore, statistical techniques should be integrated into such analysis due to the dependency of traffic analysis on statistical analysis. On the other hand, the possibility that intruders may integrate data from multiple data sources to perform successful traffic analysis should be considered.

2.3 Research into Analysis of Security Protocols

2.3.1 A Discussion of Formal Methods and Security Protocols

To begin with, it will help if we can clarify what we mean both by formal methods and security protocols and the relationship between them.

A security protocol (or cryptographic protocol) is a protocol that applies cryptographic methods to the performance of a security-related function, including key distribution, principal authentication and secure data transport over a network. A security protocol usually incorporates at least the following aspects: key agreement or establishment; entity authentication; symmetric encryption and message authentication; and reliable application-level data transport. For example, Transport Layer Security (TLS) is a cryptographic protocol employing the Diffie-Hellman key exchange that is used to secure web (HTTP) connections. It has an entity authentication mechanism, based on the X.509 system. Nevertheless, the network is usually assumed to be hostile, in that a malicious intruder may read, modify and delete transmitted messages, and may be able to impersonate or even control some principals. A robust security protocol must be able to protect messages from these malicious attacks and achieve its goals. The attacks on cryptographic protocols can be classified into two categories: intuitive and non-intuitive. The former includes the attacks dependent on properties of cryptographic algorithms, or on statistical analysis of message traffic, or on some integrations of them. The latter focuses on the attacks that are not obvious to a careful inspector. They do not intend to utilize the subtle flaws in the underlying cryptographic algorithms but simulate the above operations such as interception or masquerading in any workable sequence. With the increasing complexity of cryptographic protocols, they can sometimes be verified formally on an abstract level.

Formal methods can be viewed loosely as a combination of an abstract mathematical model of a system and corresponding requirements regarding the correctness of the system, together with a formal proof to determine the system has all the required properties that together make it functional and useful. Actually, it is unrealistic to specify formally what ‘no defect’ means for a system. All we can do is prove that the system does not contain any of

the defects that we can imagine, and that it satisfies all of the requirements. Formal verification has been widely used for cryptographic protocols since the 1980s.

There are roughly two approaches to formal verification. The first approach is logical inference. It depends on using a formal mathematical model to reason about the system, usually using theorem proving sofware such as the HOL theorem prover or Isabelle theorem prover. Usually, it is only partially automated and is driven by the user's understanding of the system to verify. The second approach is model checking, which relies on a systematic and always automatic exploration of all the mathematical model. Although they do not offer a proof of security for the entire possible state space, they provide a precise description of the conditions under which their conclusions hold, and also give an effective procedure for validating them.

Current works in formal methods mainly formalize the security problems in a discrete way. During the data exchange among a group of principals, it is easily affected by malicious attacks in a hostile open network. An intruder may perform any order of a finite collection of operations mentioned above, such as intercepting data, tampering and masquerading data, encrypting and decrypting data to make various attacks. Since many protocols are subject to such attacks and most of them are not apparent upon inspection, the formal analysis can assist us in avoiding them and ensuring the correctness of security protocols by the use of rigorous analysis of all the possible paths that an intruder could utilize.

The application of formal methods to security protocol analysis was first mentioned in the analysis of key distribution protocols for communication between two principals, such as Needham and Schroeder's public-key protocol [2]. The first work in this area was undertaken by Dolev and Yao [44] who developed a formal model where multiple executions of the protocol can be implemented concurrently. Most of work on formal analysis for security protocols is extended or adapted from this model or some variant of it. However, this still remained a fairly obscure field until the appearance of BAN logic by Burrows, Abadi, and Needham [22]. As a result, this logic was widely applied and resulted in many other logics to handle different types of problems in security protocols. Formal methods have been used for reasoning about certain security-related properties or describing and reasoning about protocols with emergent behaviour. Formal analysis for security protocols enables us to identify weakness and hidden assumptions underlying security protocols. This book aims to present fundamental techniques for formal methods in security protocol analysis and some emerging issues in this area.

2.3.2 A Brief Introduction to Protocol Abstraction

A security protocol usually uses cryptography to distribute messages, authenticate the other party and protect data over an insecure network. It can be defined as a set of transactions or traces. Each transaction consists of a series of communication events, which are perhaps some interleaved protocol runs. As described above, some fundamental security functions need to be incorporated during protocol runs. Every desirable security protocol should provide a comprehensive introduction to the details with respect to authentication, message delivery, encryption and decryption and so on. Thus, security protocols are very suitable for rigorous analytical techniques, such as inductive definitions. Theorem proving is good at defining properties of security protocols, and model checking is an effective means of detecting attacks.

Security protocols have become an inseparable part of the formal method community. They can be easily defined using semantics based on trace of events, by which it is easy to formalize and explain a security protocol. The application of formal methods to security protocol starts with the analysis of key distribution protocols for communication between two principals. Needham and Schroeder Secret-Key protocol was perhaps the first protocol to be analysed using formal methods. It is a network authentication protocol and enables principals to prove their identity to each other with the aid of a trusted server. For example, $Alice(A)$ wants to authenticate herself to $Bob(B)$ using a server S. She needs to tell the server she wants to communicate with Bob. S generates K_{AB} for the communication between $Alice$ and Bob and another key K_{BS} for $Alice$ to forward to Bob. Through two authentications that verify the key K_{AB} is held by them, $Alice$ and Bob can trust this key and use it to communicate with each other.

Two kinds of keys are applied in this protocol: private keys and long term keys. The former is generated by the trusted server S and establishes a reliable communication channel between A and B. The latter is used for communication between A and S and between B and S, respectively. The trust server needs to maintain a number of long term keys if the number of principals in a transaction is large. In that case, considerable communication and computation burdens can rest on S and consequent failure may occur at these bottlenecks. Nevertheless, the trusted server is just used when it is really necessary to establish a new key for the communication between A and B. It is unrealistic to generate new keys for each pair of principals in advance because most of them may not be used at all.

A number of cryptographic algorithms can be used for generating those keys. The strength of cryptography determines whether the keys are stronger to cryptanalyse. Nevertheless, it is not our goal to discuss the details with respect to the fundamental of cryptographic algorithms, but focus on abstract-

ing them using logical methods and verifying security protocols by rigorous reasoning.

A search of the Internet for the keyword 'security protocol', reveals a lot of security protocols designed for different purposes. They can apply different cryptographic algorithms and varied and complex authentication and authorization to set up a secure channel of communication. Regardless of these discrepancies from the details of security protocols, the abstraction of security protocols becomes less discrepant and turns out to be analogous. Thus, the relatively simple and well-known *Needham and Schroeder protocol* provides a good example to help us understand the fundamentals of security protocols. We will talk about more security protocols when discussing formal methods for security protocol analysis in the following sections.

Suppose A, B and S represent *Alice*, *Bob* and a trusted *server* as before. The protocol begins with a request from A to S, which indicates A wants to communicate with B. Logical symbol $\rightarrow$ denotes the message is transmitted from a principal(*initiator*) to another principal (*responder*).

Message 1 $A \rightarrow S$: A, B, n_A
Message 2 $S \rightarrow A$: $\{B, K_{AB}, n_A, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
Message 3 $A \rightarrow B$: $\{A, K_{AB}\}_{K_{BS}}$
Message 4 $B \rightarrow A$: $\{n_B\}_{K_{AB}}$
Message 5 $A \rightarrow B$: $\{n_B - 1\}_{K_{AB}}$

Each line in the list can be viewed as a step in a transaction, which includes the message delivered from the initiator to the claimed receiver. They are tightly correlated with each other. No matter which step fails, the transaction will not be able to be completed successfully. For convenience, we use some simple notation to denote cryptography, such as K_{AB} and K_{BS}. However, in reality, the detailed computation cannot be so simple to express. The nonces n_A and n_B denote some unpredictable number and convince *Alice* and *Bob*, respectively, that the received message is fresh. $\{X\}_K$ denotes encryption of the value X with the public key K.

In the first step, *Alice* just wants to let the server S know that she wants to communicate with *Bob*. The delivered message indicates the initiator, to whom she wants to talk to, and supplies a nonce. S generates an encrypted compound message which includes a new key K_{AB} and an encrypted message using a long term key K_{BS}. *Alice* decrypts this message using the long term key K_{AS}, and knows the contents inside. She needs to check whether the value of n_A and *Bob*'s name are consistent with the nonce and the name she sent out in the first step. The new key K_{AB} is stored for further exchanges

The message $\{A, K_{AB}\}_{K_{BS}}$ will be forwarded to *Bob* to gain his agreement on the new key K_{AB}. *Bob* decrypts the received message and knows the request is initiated by *Alice*. He uses the obtained K_{AB} to encrypt a new nonce n_B, and sends it back to *Alice* in message 4. *Alice* decrypts it to obtain

n_B. In the conventional way, she alters the value of n_B by subtracting 1 and sends it back to *Bob*, encrypted under K_{AB}.

The above informal description gives an outline of this protocol. The utility of the last two messages may, however, be unclear to some users. In messages 2 and 3, *Alice* and *Bob* know the new key K_{AB}. They actually have to believe in a remote server S because *Alice* cannot know what is inside $\{A, K_{AB}\}_{K_{BS}}$. Therefore, it is unknown whether both *Alice* and *Bob* have agreed with the common K_{AB}. It is necessary to confirm this by messages 4 and 5. They in fact assure each other that the other also knows K_{AB}.

We observe that the trusted server is critical in the protocol. S must be honest to issue the key K_{AB} for *Alice* and *Bob*, and use reliable long term keys to communicate with them. As a result, *Alice* and *Bob* will be confident that K_{AB} is supplied by the trusted server S and only for them. Furthermore, if a message encrypted under K_{AB} is received by *Alice*, it must have come from *Bob*, and vice versa. However, this protocol relies on several ideal assumptions that may give rise to subtle flaws in hostile environments. In a sense, the issues in this protocol can be used for reference in analysing other security protocols.

Regardless of the generation of many security protocols for different purposes, a number of subtle flaws have been often reported in the literature documents. To assure safe message exchanges, the protocols must be sure that intruders cannot undermine the transmitted messages by virtue of the vulnerabilities of protocols that can be exploited. This demands the correctness of protocol design and robust cryptographic algorithms. Nevertheless, here we will not discuss the details of cryptographic algorithms but just some properties of them, such as symmetric encryption and asymmetric encryption, and hash function, that can interact with security protocols. This protocol obviously depends on the assumption that *Alice* and *Bob* should be honest and never disclose their messages to a intruder or compromise their keys. Notice that if *Bob* tells the intruder his key K_{BS}, the intruder can generate a false K_{AB} and message 3. The secrecy or authentication of messages encrypted under K_{BS} will be lost. Furthermore, a more dangerous attack can be performed by the intruder in league with a certain number of dishonest principals.

In message 1, there is nothing to protect the secrecy of this message. It is possible for an intruder to know that *Alice* wants to talk to *Bob*. Also, the intruder can alter the value of n_A. As a result, the integrity of the message cannot be protected in this step. This may provide an opportunity for a denial of service because the trusted server S does not agree with the altered message by contrast with the record.

Notice that *Alice* has never explicitly declared her intention to communicate with *Bob*. Lowe [97] presents an attack on the Needham and Schroeder protocol. The attack allows an intruder to impersonate another agent *Alice* to set up a false session with *Bob*. Lowe also proposed a variation that simply includes the identifier of *Bob* so as to verify that *Bob* wishes to communicate

with *Alice*. However, encryption with *Alice*'s public key may not provide this, but is a way to provide the confidentiality service to the message sender. It seems that most analysts have assumed that inclusion of the nonce, which *Alice* sent confidentially to B, is sufficient to offer authentication. Unfortunately this is really not true, even with the most well known public key encryption algorithms.

In order to allow the receivers to authenticate the received messages, the encryption function needs to act like a message authentication code (MAC) which ensures that the message was written with knowledge of a shared secret. The above attack is probably prevented by including strict formatting in the RSA encrypted messages, such as the RSA Encryption Standard PKCS #1 [132]. Such formatting is intended to guarantee that the encrypting agent must be conscious of the whole plaintext in order to generate a valid ciphertext, such as the encryption of a shared nonce in the Needham-Schroeder protocol. Notice also that the attack is not a 'typing' attack (stream cipher) that converts ciphertext from and to plaintext, one-bit typing tags each time would still allow the attack to succeed with high probability. Thus, block cipher rather than stream cipher is more appropriate for encryption in this protocol.

The protocol designers may need to take into account more security mechanisms such as enhanced cryptographic algorithms to establish a secure communication channel between agents. Some recent developments of encryption techniques overcome some of the vulnerabilities in conventional cryptography and provide more protection on transited messages. We will explain them step by step in the following context. This protocol illustrates the general role of security protocols and their implementation in an abstracted way. It focuses on achieving authentication between two agents, but more security goals can be realized based on this protocol. From this protocol, we observe that encryption techniques play a central role in security protocols, such as encryption for secrecy or integrity, MAC for authentication, nonce for freshness, and so on. Thus, we will give an overview of the cryptographic fundamentals below.

2.3.3 A Classification of Approaches for Protocol Analysis

There have been considerable efforts in the field of the formal analysis of security protocols, as shown in Figure 2.4. Usually, the analysis consists of the following procedures:

- Formal modelling and specification of security protocols
- Specification of required security properties
- Verification of the properties
- Generation of response messages

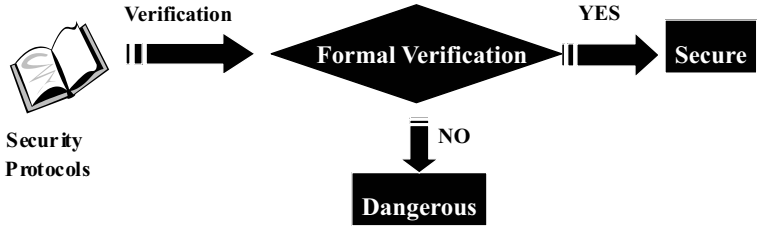

Fig. 2.4. Verification of security protocols

In the past two decades, there have been considerable efforts devoted to the formal analysis of security protocols, from which a number of approaches were extended or adapted.

Attack-Construction Methods. Dolev and Yao [44] presented a basic model for state-machine approach. Under their model an intruder is in full control of the network being able to read, modify, create, and delete messages. Effectively, the intruder is using the system being attacked as a machine to generate words (messages). The words follow some rewrite rules based, for example, on the properties of symmetric encryption. The intruder's task is to discover a word that should have been secret. Thus, the protocol security problem is transformed into a search based on a term-rewrite system. This approach was used to develop analysis algorithm for some restricted protocol classes.

Two models were then derived from the work of Dolev and Yao; the cascade protocol models, in which the users can apply cryptographic operations in several layers to form messages, and the name-stamp protocol models in which the users are allowed to append, delete, and check names encrypted together with the plaintext. The main drawbacks of the Dolev-Yao model are its failure to model the principals' ability to remember state information between states, and the fact that it can only detect protocol deficiencies [65].

Meadow's NRL Protocol Analyser [109, 111] is a prototype verification tool, written in Prolog, that aids in the verification of security properties of cryptography protocols and in the detection of security flaws. It uses the same approach as the term-rewrite model of Dolev-Yao, but treats a protocol as a machine for producing words, beliefs, and events. The NRL Protocol Analyser uses a backward search strategy to construct a path from a specified insecure state to an initial state. However, it is not easy to keep the state space workable since drastic simplifying assumptions are required.

Inference-Construction Methods (see above, Section 1.4.5). BAN logic, presented by Burrow, Abadi and Needham [22] has been widely used for the analysis of authentication protocols. BAN logic of belief belongs to the class of KD45 modal logics which in practice means that any fact is only a belief and

does not need to be universal in time and space. It assumes that authentication is a function of integrity and freshness, and uses logical rules to trace both of those attributes through the protocol. There are three phases for the analysis of a protocol using BAN logic. The first step is to express the assumptions and goals as statements in a symbolic notation so that the logic can proceed from a known state to one where it can ascertain whether the goals are in fact reached. The second step is to transform the protocol steps into symbolic notations. Finally, a set of deduction rules called postulates are applied. The postulates should lead from the assumptions, via intermediate formulae, to the authentication goals.

BAN logic has been a success. It has found flaws from several protocols, such as Needham-Schroeder [120] and CCITT X.509 [79]. It has uncovered redundancies in many protocols, such as Kerberos [116] and Otway-Rees [125]. Many papers make claims about their protocol's security based on this logic.

Inevitably, criticisms on various features of the BAN logic have been published. According to Liebl, it is difficult to prove properties of the BAN logic, such as completeness, and the logic does not take into account the release of message contents and the interaction of the runs at different times of the same protocol [94]. Thus, a successful but rather complicated approach called GNY logic [60, 62] was proposed, which increases the range of BAN logic.

GNY logic aims to analyse a protocol step-by-step, making explicit any assumptions required, and drawing conclusions about the final position it attains. This logic offers important advantages over BAN logic. The GNY approach [63] places a strong emphasis on the separation between the content and the meaning of messages, which increases consistency in the analysis and introduces the ability to reason at more than one level. In GNY logic, principals can include the message data which they do not believe in. However, GNY logic addresses only authentication and is much more complicated and elaborate than other methods as it has many rules which have to be considered at each stage [65].

BGNY logic, introduced by Brackin [14] is an extended version of GNY. This belief logic is used by software that automatically proves the authentication properties of cryptographic protocols. Similarly to GNY logic, BGNY addresses only authentication. However, BGNY extends the GNY logic by including the ability to specify protocol properties at intermediate stages and being able to specify protocols that use multiple encryption and hash operations, message authentication codes, hash codes as keys, and key-exchange algorithms.

SvO [150], presented by Syverson and van Oorschot is designed to capture the features of extensions and variants of four logics, namely BAN, GNY, AT [2], and vO, in a single unified framework. In addition, the authors provide model-theoretic semantics with respect to which the logic is sound. The SvO logic was intended to encompass the reasoning of these other logics while

providing a rigorous understanding of its formal expressions. The SvO logic is considered to be simpler to use and more expressive than any of the logics from which it is derived.

In addition, Kailar [82] has proposed a special purpose logic, which is used for the analysis of secure e-commerce protocols that require accountability. This logic is more suitable for the analysis of accountability than the belief logics. In the same framework, an authentication logic presented by Kessler and Neumann [85], based on the AUTLOG [86] semantics, can analyse the accountability of transactions in the framework of electronic commerce protocols.

Proof-Construction Methods. As mentioned above, inference-construction methods do not address secrecy, often lack clear semantics, and it is sometimes difficult to say exactly what a belief-logic proof actually proves. On the other hand, attack-construction methods may have to search spaces that grow exponentially with the size of the protocol, so the time and space they require can easily exceed all available resources.

In order to confront these shortcomings, Bolignano [9] has proposed an approach targeting the generation of human-readable proofs. To achieve this goal, specific properties of the problem are used to formalize the requirements and simplify the proofs.

Paulson [128, 129] has independently developed a similar approach synthesizing the inference-construction method idea of protocol message guarantees and the attack-construction method notion of event. Paulson defines protocols inductively as the set of all possible event traces. This approach allows the modelling of both attacks and key losses.

Within the same framework, Schneider presents a general approach for the analysis and verification of authentication properties in the language of CSP [137]. The focus of this research work is the development of a specific theory targeted towards the analysis of authentication protocols and built on top of the general CSP semantic framework.

Fabrega, Herzog, and Guttman introduced the notion of strand space [49, 50]. They presented a model and a set of proof methods for cryptographic protocols along the line of the NRL Protocol Analyser, Schneider's work, and Paulson's inductive definitions. In conjunction with the aforementioned formalism the authors use the concept of ideals to prove bounds on a penetrator's capabilities independently of the security protocol being analysed.

Other Related Approaches. Recently, *proof-construction approaches* have been developed to avoid the exponential searches of *attack-construction approaches* by replacing them with theorems regarding these searches. Also, this method complements the *Inference-construction methods* because it is based on the problem formalization via hypotheses and authentication properties, but relies on problem-specific properties and a specification at a bet-

ter level of precision. *Proof-construction methods* formally model the actual computations performed in protocols and prove theorems about these computations. The representative works were presented by Bolignano and Paulson [9, 15, 128, 129], and Brakin [17].

The above techniques mainly focus on the analysis of cryptography protocols and authentication protocols. There have been attempts to model more realistic protocols, such as the protocols developed by CyberCash. Meadows and Syverson [113] have designed a language for describing the SET specifications [138, 139, 140], but have left the actual analysis to future researchers. Bolignano [10] has applied his approach to analyse some cases of electronic payment protocols, such as C-SET and SET.

Limitations of Existing Approaches. As already described, a number of methods have been developed for the analysis of security protocols. However, some subtle flaws can still be found from actual transactions [97, 98, 120] owing to inherent limitations in existing formal methods.

1. The attack-construction approaches suffer from a big state space and are inefficient in detecting attacks on complicated secure transaction protocols.
2. The inference-construction approaches are still too complicated to verify security protocols. For example, BAN logic requires a large number of assumptions where a secret remains secret during the execution of protocols; and the GNY logic contains many rules that have to be considered at each stage.

Other works emphasize a particular aspect of security protocols. Among them, attempts to verify e-commerce protocols are not as mature as those used for authentication protocols and cryptography protocols. Moreover, the aforementioned methods have been incompetent to deal with emerging issues of security protocols, such as the properties of fairness and liveness in financial transactions, and new type of threats, such as denial of service, traffic analysis and collusion attack. Moreover, it is not easy to handle the problems using the conventional analysis because they are varied, complex, and sometimes even covert. To our knowledge, existing methods cannot cover this problem in many cases.

In contrast to the above methods, in this book we propose ENDL, which overcomes some of the limitations. It considers the possibilities of 'message lost' and 'host crash' environment. Furthermore, ENDL is able to deal with the practical financial circumstances and accommodate some potential attacks such as replay attacks and collusion attacks.

The existing verifications were mainly implemented by theorem proving. Recently, a number of works have been put forward for automating the verification of security protocols.

2.4 Attack-Construction Approach

This approach focuses on developing a formal model based on the algebraic term-rewriting properties of cryptographic systems. It was introduced by Dolev and Yao [44], from which a number of relevant works were developed, either extending it or applying the same concept to varied types of issues in cryptographic protocols, such as the work of Syverson [149], Meadows [109] and Woo & Lam [160]. Automated analysis has been developed in the recent application of this approach. Thus, a user is able to query the system for known attacks.

This approach usually involves an analysis of the reachability of certain system states. In this regard, it is similar to the approaches that aim to develop expert systems. However, this approach tries to present that an insecure state cannot be reached, but the expert systems start with an insecure state and attempt to show that no path to that state could have originated at an initial state.

2.4.1 Approaches by Dolev and Yao

There have been considerable efforts to develop public key encryption to guarantee secure network communication. Although they are effective against the passive eavesdropper, an improperly designed protocol could be vulnerable to an active attacker, one who may impersonate another user and may modify or replay the message [44]. This issue was pointed out by Needham and Schroeder, in what is probably the first mention of formal methods as a possible tool for cryptographic protocol analysis [120]. However, the first work that was practically done in this area was addressed by Dolev and Yao, and slightly later by Dolev, Even and Karp [43], who developed a set of polynomial-time algorithms for determining the security of a restricted class of protocols.

Dolev and Yao proposed the first algebraic model for the security of properties. They define the precise mathematical models according to the basic assumption on the system that they want to model. In the Dolev-Yao term-rewriting model [44], it is assumed that there is an intruder who is able to read all message traffic, modify and destroy any message traffic, and perform any operation (such as encryption or decryption) that is available to a legitimate user of the protocol. It is assumed that there is some set of words (messages), such as encryption keys, possessed by honest principals, that have been encrypted, that is unknown to the intruder. The intruder can use the system being attacked as a machine to generate words. The intruder's goal is to discover a word that should have been secret. The protocol can be regarded as an algebraic system manipulated by the intruder because any message received by an honest principal can be thought of sending by the intruder.

1. In a perfect public key system
   - the one-way functions used are unbreakable;
   - the public directory is secure and cannot be tampered with;
   - everybody can gain access to all the encryption functions E_x;
   - only X knows decryption function D_x.
2. In a two-party protocol, only two parties who wish to communicate are involved in the distribution of secrets; the third party for authentication in encryption or decryption is unnecessary.
3. In a uniform protocol, the coherent format is applied by each pair of principals who wish to communicate.
4. Assumptions with respect to an attacker:
   - He can obtain any message passing through the network.
   - He is a legitimate user and thus able to initiate a communication with any other user.
   - Any user A may have the opportunity to become a receiver to any other user B.

Dolev and Yao modelled some classes of protocols, reasoned about these protocols and proved some interesting properties of these protocols. Two models were developed.

1. Cascade protocol model, in which the users can apply only the public key encryption-decryption operations (E_X,D_X) to form messages, in which X is a user name. several layers of these operators may be applied.
2. Name-stamp protocol model, in which the users are additionally allowed to append (i_X), delete (d), and check (d_X) names encrypted together with the plaintext. Thus, Name-Stamp Protocols are a generalization of the above Cascade Protocols.

Cascade protocol. Operators for Cascade Protocols consist of encryption E_X and decryption D_X. Suppose X and Y are two parties in the protocol. We have

- X operates strings of E_X, E_Y and D_X.
- Y operates strings of E_X, E_Y and D_Y.

As a consequence, a protocol can be represented by a series of strings of which X operates O_X and of which Y operates O_Y. Thus, a cascade protocol can be defined as

1. $X \rightarrow Y$: $<O_X^1>M$
2. $Y \rightarrow X$: $<O_Y^1><O_X^1>M$
3. $X \rightarrow Y$: $<O_X^2><O_Y^1><O_X^1>M$

The strings can be continuously operated by X and Y. Let γ be a string of operators over encryption function E and decryption function$D$ and let $\overline{\gamma}$ be

the reduced form of γ after applying algebraic laws $E_X D_X = D_X E_X = 1$ in terms of the public key cryptosystem.

To illustrate this, Let $O_X = E_Y D_X$ be the strings that X operates, and let $O_Y = E_X D_Y E_X E_X D_Y$ be the strings that Y operates. The protocol is defined as.

1. $X \rightarrow Y$: $E_Y D_X M$
2. $Y \rightarrow X$: $E_X D_Y E_X E_X D_Y M$

Let $\overline{N(X,Y)}$ be the notation of reduced form in terms of the algebraic law mentioned above. After reduction, we have $\overline{N_1(X,Y)} = E_Y D_X$ and $\overline{N_2(X,Y)} = E_X D_Y E_X$.

Let Σ be the alphabet of operator symbols and $\Sigma^\star$ be the set of all strings over Σ including the empty word λ. Let Z be an adversary (or intruder), let $\Sigma(Z) = E \cup \{D_Z\}$, let $\Sigma(X)$ be all strings which X operates, and let $\Sigma(Y)$ be all strings which Y operates. Based on the above definition, a cascade protocol can be determined as secure or insecure according to the following conditions.

- A protocol is insecure if there exists some string γ in $\{\Sigma(Z) \cup \Sigma(X) \cup \Sigma(Y)\}^\star$ such that for some $N_i(X,\ Y)$, $\overline{\gamma N_i(X,\ Y)} = \lambda$;
- A protocol is secure, otherwise.

We can use one example to illustrate this, in which two parties A and B are included. Usually, the protocol is defined as.

1. $A \rightarrow B$: $(A,\ E_B(M),\ B)$
2. $B \rightarrow A$: $(B,\ E_A(M),\ A)$

Usually, the operations involved can be represented as $E_X D_Y E_Y M$ without intruder. However, the protocol can be broken by an intruder Z.

1. $A \rightarrow Z(B)$: $(A,\ E_B(M),\ B)$
2. $Z \rightarrow B$: $(Z,\ E_B(M),\ B)$
3. $B \rightarrow Z$: $(B,\ E_Z(M),\ Z)$
4. Z knows M by decrypting $E_Z(M)$

The intruder Z intercepts a message from A at the beginning, and pretends to be A to send the message to B. B will think this message is from A, and will send a message back to Z. This procedure can be represented as $D_Z E_Z D_Y E_Y M$ due to the intruder Z.

Name-stamp protocol. Operators for Name-Stamp Protocols are composed of the following operations:

- encryption E_X
- decryption D_X
- append i_X

- name-match d_X
- delete d.

Based on the properties of the public key cryptosystem and the name-stamps we have the following algebraic laws of the reduction/rewriting system:

- $E_X D_X = D_X E_X = 1$
- $d_X i_X = d i_X = 1$

Suppose a, b and c are three elements of Σ. From the observation, if $ab = bc = 1$, then $a = c$. Let a string $\gamma = head(\gamma)tail(\gamma)$, in which $tail(\gamma)$ represents a suffix of n bits. Thus, we have

- $i_X \gamma = \gamma X$
- $d_X \gamma = head(\gamma)$ if $tail(\gamma) = X$
- $d\gamma = head(\gamma)$

In particular, there is a constraint on the d_X operation. That is, when d_X is applied, the transmission will not proceed unless the identity of user X is confirmed, namely $tail(\gamma) = X$. Consequently, for any text γ transmitted between normal users X and Y, there should be no d_X or d_Y remaining after the algebraic laws are applied.

Let Z be an adversary (or intruder), let $\Sigma(Z) = \{D_Z\} \cup \{I_A, E_A, D_A, d \text{ all } A\}$, let $\Sigma(X)$ be all strings which X operates including new operations, and let $\Sigma(Y)$ be all strings which Y operates including new operations. In the same way, we can decide whether a name-stamp protocol is secure or insecure in terms of the following conditions.

- A protocol is insecure if there exists some string γ in $\{\Sigma(Z) \cup \Sigma(X) \cup \Sigma(Y)\}^\star$ such that for some , $\overline{\gamma N_i(X, Y)} = \lambda$;
- A protocol is secure, otherwise.

To illustrate this, we use a brief example including two parties A and B. The protocol is defined as.

1. $A \rightarrow B$: $(A, E_B(E_B(M)A), B)$
2. $B \rightarrow B$: $(B, E_A(E_A(M)B), A)$

Suppose the normal operations by X and Y without intruder and the operations by X and Y along with intruder Z are represented by $\alpha(X, Y)$ and $\alpha_Z(X, Y)$, respectively. We have

1. $\alpha(X, Y) = \{E_Y i_X E_Y\}$
2. $\alpha_Z(X, Y) = \{E_X i_Y E_X D_Y d_X D_Y\}$

Thus, we can obtain two corresponding reduced forms in terms of the algebraic laws mentioned above.

1. $\overline{N_1(X,Y)} = E_Y i_X E_Y$
2. $\overline{N_2(X,Y)} = E_Y i_Y E_Y$

Let L be the context-free language of all possible protocol traces of the *Ping-Pong* protocol that can be reduced by the cancellation rules to the empty word λ. Let L_Z be the regular language of all possible attacks. In essence, the security problem of the *Ping-Pong* protocol can be transferred to the question of whether the intersection of a regular language L_Z and a context-free language L is non-empty, namely $L_Z \cap L_Z \neq \emptyset$. For the Ping-Pong" protocol by Dolev, Even and Karp, there exists a security checking algorithm whose input are the generic cancellation rules and the protocol. The time complexity is $O(n^3)$, in which n is the length of the input, and the space complexity is $O(n^2 m)$, in which m is the length of the context-free grammar G such that $L = L(G)$ and n is the number of states of the non-deterministic finite automaton A such that $L_Z = L(\mathrm{A})$.

However, it was soon discovered that loosening the restriction on the protocols even slightly can make the security problem undecidable, which prevents the work from going further. Regardless of its limitation, Dolev and Yao's work was still significant and enables the definition of a formal model of an environment where a set of protocols instead of an individual protocol can be running simultaneously. Most of the later work on the formal analysis of cryptographic protocols has been developed from this method.

2.4.2 NRL Protocol Analyser

Meadows's NRL Protocol Analyser [109, 111] uses the same term rewriting properties of protocol specifications as Dolev-Yao, and is able to assist either in the verification of security properties of cryptographic protocols or in the detection of security flaws. Unlike the Dolev-Yao model, which treats a protocol as a machine for producing words, the NRL Protocol Analyser treats a protocol as a machine for producing not only words, but also beliefs and events. Each protocol participant includes a set of beliefs in a NRL model. These beliefs are generated or altered as the result of receiving messages composed of words, while messages are sent based on both beliefs and messages received. Events show the state transitions in which new words are created and beliefs are altered. Consequently an intruder who controls the dissemination of messages can use the protocol to generate words, beliefs, and events.

The NRL Protocol Analyser has been successfully applied to locate a series of previously unknown flaws in a number of protocols [21, 145], and to demonstrate flaws that were already known in the literature [84]. It uses a backward search strategy to construct a path from a specified insecure state to an initial state. The NRL model aims to prove that a protocol is secure by constructing a single path using an arbitrary number of protocol rounds

thereby working in an infinite state space. The main drawbacks of the current implementation are listed below

- Some drastic simplifying assumptions to keep the state space workable.
- As with most rule-rewrite systems, it is not clear how well the system adjusts as more complicated algorithms will need to be expressed by an increasing set of rules.
- The generation of lemmas stating that infinite classes of states are unreachable: these have to be proved by hand.

Syverson and Meadows then defined a formal language for specifying and reasoning about cryptographic protocol requirements by developing the NRL protocol analyser [149]. The analyser is used as a model checker to assess the validity of the formulae that make up the requirements. In this formal language, three specific notations are applied.

- $\rightarrow$ represents the standard condition.
- $\wedge$ represents conjunction.
- $\diamond$ represents a temporal operator meaning *at some point in the past.*

It assumes that principals can keep track of rounds of protocols from their perspective via local round numbers. The following actions are defined.

- $accept(B, A, Mes, N)$ indicates that B accepts the message Mes as from A during B's local round N.
- $learn(Z, Mes)$ indicates the intruder Z learns the word Mes.
- $send(A, B, (Query, Mes))$ indicates that the A sends B Mes in response to query $Query$.
- $request(B, A, Query, N)$ indicates B sends query $Query$ to A.

Based on the above constructs and actions, the requirements can be defined by a conjunction of statements. This will facilitate the application of the NRL Protocol Analyser. For example:

Requirement 1

- $\neg(\diamond \text{accept}(B, A, Mes) \wedge learn(Z, Mes))$
- $accept(B, A, Mes, N) \rightarrow$
  $\diamond send(A, B, (Query, Mes)) \wedge$
  $\diamond request(B, A, Query, N))$

Requirement 1 contains two conditions, and both of them must hold. The former represents that B accepted message Mes from A in the past and the intruder Z did not learn Mes in the past. The latter represents that if B accepted message Mes from A in B's local round N, then A sent Mes from A to B in answer to a query at B's local round N.

In some cases, it is perhaps unnecessary that A sends the message in response to B's query, only after B's query. The relaxation of this requirement can be captured by the omission of the *Query* from the *send* and *request* actions. Thus we have

Requirement 2

- $\neg(\diamond\text{accept}(B, A, Mes) \wedge learn(Z, Mes))$
- $accept(B, A, Mes, N) \rightarrow$
  $\diamond send(A, B, (Mes)) \wedge$
  $\diamond request(B, A, N))$

Alternatively, it may be required that the messages from A and B are recent. In that case, B will not accept a message that arrives too late after he requests it or after A sends it. This can be defined by the following requirement.

Requirement 3

- $\neg(\diamond\text{accept}(B, A, Mes) \wedge learn(Z, Mes))$
- $accept(B, A, Mes, N) \rightarrow$
  $\diamond send(A, B, (Mes)) \wedge \diamond request(B, A, N)$
- $accept(B, A, Mes, N) \rightarrow$
  $\diamond send(A, B, (Mes)) \wedge \diamond request(B, A, N)$
- $accept(B, A, Mes, N) \rightarrow$
  $\diamond send(A, B, (Mes)) \wedge \neg(\diamond time\_out(B, N)) \wedge$
  $\neg(\diamond time\_out(A, N))$

The notation *time_out* is used to control the currency of messages. The NRL distinguishes the honest principal from dishonest principals. Correspondingly, an honest principal and a dishonest principal can be represented by $user(A, honest)$ and $user(A, dishonest)$, respectively. And a user who may be honest or dishonest can be represented by $user(A, Y)$, in which Y means a variable. Syverson and Meadow provide a complicated requirement to represent that an honest B accept a message as coming from an honest user A only if it was never previously accepted by an honest user.

Requirement 4

- $\neg\diamond accept(user(B, honest), user(A, honest), Mes) \vee \neg\diamond learn(Z, Mes)$
- $accept(user(B, honest), user(A, honest), Mes, N) \rightarrow \diamond send(user(A, honest), user(B, honest), Mes) \wedge\diamond\ request(user(B, honest), user(A, honest), N)$
- $accept(user(B, honest), user(A, honest), Mes) \rightarrow$
  $\neg\diamond(accept(user(C, (honest)), user(D, Y), Mes))$

The variable Y means that the message must not have been previously accepted by any user irrespective of his honesty.

The actions in the requirement are actually transitions from one state to another. This can be represented by ordered pairs of the state of the form (s, s'), where s indicates the state before the action and s' indicates the state after the action. In particular, s_t means the state at the time t. A state space S consists of a set of states, and a *trace* σ is a sequence of elements of S.

A model is applied to define the satisfaction relationship between a formula and a quadruple, $<S, I, \sigma, t>$, in which I is an interpretation of atomic formulas. Suppose α is a formula. $<S, I, \sigma, t> \models \alpha$ represents that α is true at $<S, I, \sigma, t>$, in which $\models$ is the relationship between models and formulae. More details of this satisfaction relationship can be found in the paper [149].

The model used by NRL is an extension of the Dolev-Yao model. It is assumed that an intruder has the ability to read all message traffic, destroy and alter messages, and create his own message. Dishonest users are modelled as intruders. Thus, they are not specified separately. The functions *lfact*() represents a set of learned facts, and *intruderknows*() represents the *lfacts*() known by the intruder.

Let [ ] be empty list. Suppose the action (s, s') represents user A attempting to initiate a communication with B during local run N at time t. We will obtain different results in state s and in state s'.

- in state s, *lfact*(*user*(A, *honest*), N, *init_comm*, t) = [ ]; and
- in state s', *lfact*(*user*(A, *honest*), N, *init_comm*, t+1) = [*user*(B)]

Certainly, the value of *lfact* would also be [ ] at any time before t.

Similarly, two actions are associated with intruder knowledge. Let W be a message and let (s, s') be ordered pairs of state as above. Suppose A sends a message W to B at time t_1, and the intruder intercepts W at time t_2. In the similar way, the results of *lfact* in different states can be represented as

- in state s, *lfact*(*user*(A), N, *send_to_B*, t) = [ ]; and
- in state s', *lfact*(*user*(A), N, *send_to_B*, t+1) = [W]

The intruder gains knowledge in two ways. One way is given by the change of (s', s''), and the second way is to do some available internal operations on things he already knows. Let w be some n-ary operations that the intruder can do. Thus we have

1. in state s'', *intruderknows*(t_1) = *intruderknows*(t_2) $\cup$ {[W]}; and
2. *intruderknows*(t_2) = *intruderknows*(t_1) $\cup$ {$w(W_1, \cdots, W_n)$}

where t_1 and t_2 are global times and correspond to A's local time t and t+1, respectively. *intruderknows*() can be viewed as the *lfact*() learned by the intruder.

Thus, a protocol can be defined by the above actions and constructs. Then, a set of *lfact* S is defined, in which S can include a potential attack by an intruder. The analyser then decides whether the set S can meet the requirements of the protocol. If so, it indicates a possible attack is identified.

A specification in the NRL analyser contains four steps. The first step defines the transition rules for the actions of honest principals. The second step specifies the operations that are available to the honest principals and possibly to the intruder, such as encryption and decryption. The third step defines the atoms that are used as the basic building blocks of the words in the protocol. The last step defines the rewrite rules obeyed by the operations.

A transition rule consist of three parts. The first part describes the conditions that must hold before the rule can fire, including the words the intruder must know, the values of the available *lfacts* and any constraints on the *lfacts* and words. The second part gives the conditions that hold after the rule fires according to the words learned by the intruder and any new values by *lfacts*. The final part describes an event statement, including the name of relevant principal, the number of the protocol round, the event and the value of the principal's counter. An example of transition rule is:

If:
$count(user(B,\ honest)) = [M]$,
$lfact(user(B,\ honest),\ N,\ recwho,\ M) = [user(A,\ Y)]$,
$not(user(A,\ Y) = user(B,\ honest))$,
then:
$count(user(B,\ honest)) = [s(M)]$,
$intruderlearns([user(B,\ honest),$
$rand(user(B,\ honest),\ M)]$,
$lfact(user(B,\ honest),\ N,\ recsendsnonce,\ s(M)) =$
$rand(user(B,\ honest),\ M)]$
EVENT :
$event(user(B,\ honest),\ N,\ requestedmessage,\ s(M)) =$
$[user(A,\ Y),\ rand(user(B,\ honest),\ M)]$.

This rule describes the sending of a nonce from an honest principal B to another principal A whose honesty is unknown. In this rule, there are two *lfacts* including *recwho* and *recsendsnonce*. The former is used to keep the name of the user B is communicating with, and the latter indicates the random nonce that B sends to A. Eventually, the event statement *requestedmessage* keeps the words used in this rule, including the name of user A and the random nonce.

In addition to transition rules, the rewrite rules are also defined to reduce words to simple words. An example of a rewrite rule that describes the encryption with public key and private keys, respectively.

- $\text{pke}(\text{privkey}(X),\ \text{pke}(\text{pubkey}(X),\ Y)) \Rightarrow Y$
- $\text{pke}(\text{pubkey}(X),\ \text{pke}(\text{privkey}(X),\ Y)) \Rightarrow Y$

It is the purpose of the NRL Protocol Analyzer to show that a given protocol specification meets its requirements. However, this cannot ensure that the protocol is secure. This analyser thus can be used as a tool and combined with other tools, to constitute a proof that the protocol is secure.

2.5 Inference-Construction Approach

Most of the early work based on the Dolev-Yao model or some variant used some types of state exploration technique, in which a state space is specified and then explored by the tool to determine if there are any paths via the space corresponding to a successful attack by the intruder.

2.5.1 BAN Logic

BAN logic, developed by Burrows, Abadi and Needham [22], uses a different approach from that of the state exploration tools. It is a logic of belief, which is composed of a set of modal operators describing the relationship of principals to data, a set of possible beliefs that can be held by principals, and a set of inference rules for inferring new beliefs from old ones.

In the BAN logic of belief, any fact is usually regarded as a belief. Authentication is assumed to be a function of integrity and freshness. It uses logical rules to trace both of those attributes through the protocol. The analysis of a protocol by BAN logic usually consists of three main steps.

- Firstly, the assumptions and goals are expressed as statements so that the logic can proceed from a known state to one and see whether the goals are in fact reached.
- Secondly, the protocol processes are transferred into symbolic notation.
- Finally, a collection of inference rules are applied. The inference rules should lead from the assumptions and intermediate formulae, to the authentication goals.

Intuitively, there are the following assumptions in the logic.

- If A has sent B a message m_1 that A has never used for this purpose before and if A subsequently receives another message m_2 that depends on knowing m_1, then A believes that m_2 is recently originated by B.
- If A believes that a key K is shared only by him and B, and A sees a message m encrypted with K, then A believes that message m comes from B to A.
- If A believes that K is B's public key, then A should believe that any message can be decrypted with K comes from B.
- If A believes that only A and B know m, then A should believe that any encrypted message A receives containing m comes from B.

The only propositional connective is conjunction, denoted by a comma. In the logic, conjunction is treated as sets with properties such as associativity and commutativity. In addition to conjunction, the following constructs are used in the logic.

- P **believes** X. P believes X or P is entitled to believe X.
- P **sees** X. A message containing X was sent to P, who can read and repeat X.
- P **said** X. At some time, P sent a message that includes X, but it is uncertain when the message was sent.
- P **controls** X. P is an authority on X and should be trusted on this matter. An example is a server which is often trusted to generate encryption keys properly.
- **Fresh**(X). That is, X has not been sent in a message at any time before the current run of the protocol. This is defined to be true for nonces, that is, expressions generated for the purpose of being fresh.
- $P \overset{K}{\leftrightarrow} Q$. P and Q may use the shared key K to communicate. The key K will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.
- $\overset{K}{\mapsto} P$. P has K as a public key. The matching secret key, K^{-1}, will never be known by any principal other than P or a principal trusted by P.
- $P \overset{K}{\leftrightarrows} Q$. The formula X is a secret known only to P and Q, and the principal they trust. X can be used to prove their identities to one another.
- $\{X\}_K$. This represents the formula X encrypted under the key K. The originator of X is often omitted, and it is assumed that each principal is able to recognize and ignore his own messages.

Logical inference rules are formed from these basic constructs. The BAN logic is composed of a very concise, intuitive set of rules, which made it easy to use. Even so, as the BAN paper demonstrated, it was possible to use the logic to detect serious flaws in protocols. The main logical inference rules used in the logic are listed below.

1. The *message-meaning* rules. This rule is used to derive beliefs about the origin of messages. For shared keys, the rule is described as

$$\frac{P \ believes \ P \overset{K}{\leftrightarrow} Q, P \ sees \ \{X\}_K}{P \ believes \ Q \ said \ X}$$

   This represents that if P believes the encrypted key K is shared between Q and P and sees X is encrypted by K, then P believes that Q once said X. It assumes that P did not send the encrypted message. Thus, if the originator of this message is from R, it requires $R \neq P$. In the similar manner, the logic defines the inference rules with respect to public keys

and shared secrets. All of them actually attempt to interpret different kinds of messages and derive beliefs about the message origin from them.

2. The *nonce-verification* rule. It represents that a message is recent and still believed by the sender himself.

$$\frac{P \ believes \ fresh(X), P \ believes \ Q \ said \ X}{P \ believes \ Q \ believes \ X}$$

That means that if P believes X was uttered in the present and that Q once said X, then P believes that Q believes X. This rule reflects, in an abbreviated way, the practice of using challenges and responses for authentication. Nevertheless, X must be plaintext rather than cipherext.

3. The *jurisdiction* rule. This shows that if P believes Q has authority over X then P believes Q in the truth of X.

$$\frac{P \ believes \ Q \ controls \ X, P \ believes \ Q \ believes \ X}{P \ believes \ X}$$

4. The *see* rule. If a principal P sees a formula, then he should also see its components. In some cases, the necessary key is required. An example is given below, and many other such rules can be see in BAN logic.

$$\frac{P \ sees(X, Y)}{P \ sees \ X}$$

5. The *fresh* rule. If the formula X is fresh, then any formula that contains X is fresh.

$$\frac{P \ believes \ fresh(X)}{P \ believes \ fresh(X, Y)}$$

Idealized protocol. A security protocol can be idealized, according to these inference rules. In contrast to the informal protocol description that is often ambitious, the logic is more appropriate for formal analysis. Therefore, we can transfer each protocol step into an idealized form. For example,

$$A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}$$

In this message in informal form, A tells B who knows the key K_{bs}, that K_{ab} is the key to communicate with A. This step can be idealized as

$$A \rightarrow B : \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

In the idealized form, some cleartext messages are removed as they can be intercepted by intruders and do not contribute to the beliefs of the recipient.

BAN logic has been successfully used to discover flaws or redundancies in many protocols, such as Needham-Schroeder [120] and Kerberos [87]. Nevertheless, some criticisms have been published on certain properties in the

logic, such as informality in its operational semantics [147]. Nessett criticized BAN logic over its claimed goals of authentication [121], but Syverson claims that this criticism is invalid because BAN does not claim to provide security, but trust. The most criticized points are the logic's incomplete semantics and modelling of freshness, which can result in ambiguity and vagueness at the idealization step. Eventually, the debate has led to a clearer understanding of the role of knowledge and belief in the analysis of key management schemes. It is required to define specifically the relationship between belief and knowledge.

Therefore, many other logics are developed from BAN logic. They either extend or adapt BAN logic or use the same concept for different types of problems in cryptographic protocols.

2.5.2 Extensions to BAN Logic

The GNY logic is a successful but complicated method [62]. This logic aims to analyse a protocol by making explicit any required assumptions and drawing conclusions about the final position it reaches. It separates the content from the meaning of messages to enhance the consistency in the formal analysis.

An important notion presented in the logic is *recognizability* to represent the fact that a principal expects certain formats in the messages it receives. Every principal in a protocol has specific expectations about the message he or she will receive. For example, if it is specified that A will receive nonces N_a and N_b in a protocol step, then the subsequent N_a and N_b will be viewed as nonces.

Another important contribution of the GNY logic is the distinction of belief from possession. Consequently, each principal has a *belief set* and a *possession set*. In addition, GNY explicitly represents whether a principal generated a message itself. Together with basic constructs of BAN, the following are included in GNY logic.

- $P \triangleleft X$. The principal P is told formula X. P receives X after performing some computations, such as decryption.
- $P \ni X$. P *possesses*, or is capable of possessing, formula X. This includes all formulae that P has been told, that P generated or that are computable from the formulae he already possesses.
- $P \mid\equiv \phi(X)$. P *believes*, or is entitled to believe, that formula X is recognizable and capable of possessing, formula X. This includes all formulae that P has been told, that P generated or that are computable from the formulae he already possesses.

In particular, GNY logic defines a *not-originated-here* formula by prefixing a star to the formula. For example, $P \triangleleft *X$ represents that P is told a formula which he did not convey previously in this protocol run.

Based on the constructs, the logic also defines inference rules like BAN logic. Several representative rules are described below.

- *Being told rule.*

$$\frac{P \ \triangleleft \ (X, Y)}{P \ \triangleleft \ X}$$

If P is told a formula, then P is told each of its concatenated components. A similar rule can be that the told formula is encrypted by a key K that is known by P, then P is told the decrypted contents of that formula.

- *Possession rule.*

$$\frac{P \ \triangleleft \ X}{P \ \ni \ X}$$

The principal P is capable of possessing anything he is told. Furthermore, if P possesses more than one formula, he possesses the concatenation of them.

- *Recognizability rules.*

$$\frac{P| \equiv X}{P| \equiv \phi(X, Y), P| \equiv \phi(F(X))}$$

That means if a formula is recognizable by P, then any formula containing the formula and the computational function of this formula are recognizable.

$$\frac{P| \equiv X, P \ni K}{P| \equiv \phi(\{X\}_K), P| \equiv \phi(\{X\}_K^{-1})}$$

That means if a formula is recognizable and the key K is possessed by P, then the encryption and decryption of the formula by K are recognizable.

- *Rationality rule.*
Another important supplement in GNY logic is the rationality rule. This rule shows that the current set of rules can be extended to allow principals to derive rational conclusions about the state of other principals. For example, if we have a postulate.

$$\frac{C_1}{C_2}$$

then we can obtain a new postulate below:

$$\frac{P| \equiv C_1}{P| \equiv C_2}$$

The GNY logic is applied to detect the flaws in the Needham and Schroeder protocol. In comparison with BNY logic, it separates the content from the meaning of a message. The logic has no the universal assumption that all principals are honest and competent, and reasons about beliefs held by others in terms of trust of different levels.

GNY logic however is more complicated and elaborate than other approaches owing to the many inference rules needed to be considered at each stage. As a result, there have been a number of extensions of the logic to accommodate to other protocols. For example, BGNY logic [14] is an extended version based on higher order logic (HOL) theory. It extends the GNY logic by including the ability to specify protocol properties at intermediate stages and being able to specify protocols that use different operations such as encryption and hashing operations.

Extending BAN by adding probability reasoning. In [24], Campbell et al. claim that BAN logic is only able to evaluate the trust that can be put on the goal by the legitimate communicants using beliefs of the principals. However, it is unable to model insecure communication channels or untrustworthy principals. They extend BAN logic to a probabilistic logic along with the added capability of modelling hostile environments and drawing a conclusion in such cases.

The beliefs of principals are quantified by attaching probability values to the corresponding statements in the logic. The probability of a belief and the probability of an inference rule are defined as the partial belief and the probability that the rule holds. It aims to compute the probability that the proof is valid or justifiable in terms of the probabilities of the assumptions of the proof.

Let $\mathcal{B}$ be the union of the set of axioms and the set of rule instances and let w be an interpretation. Thus, the probability of $\alpha \in \mathcal{B}$ is defined as

$$P(\alpha) = P(\{w : w(\alpha) = 1\})$$

Let c be a conclusion which can inferred from $\mathcal{B}$. The interpretation of w is extended to c by defining $w(c) = 1$ only if there is a proof c from $\mathcal{B}$. In a similar way, the probability of a conclusion c can be defined as

$$P(c) = P(\{w : w(c) = 1\})$$

In each state of a protocol run, only some of the sentences and some of the inference rules can be valid. Thus, only a subset of the conclusions are possible. Therefore, the probability of c can be viewed as the probability that the formal proof of c is valid and depends on the probabilities that the assumptions and rules are valid.

There may be more than one way to infer the conclusion c by the given probability from $P(a_i)$ $(i = 1, \cdots, n)$. A linear programming of the problems

makes it possible to find an interval $[L, U]$ such that $L \leq P(c) \leq U$. Let p_1, $\cdots$, p_n be an assignment of probabilities to the assumptions $a_1, \cdots, a_n$ of a proof of the conclusion c. Then, the lower limit L (respectively upper limit U) can be obtained by the simplex linear algorithm:

$$\begin{aligned} &\textit{minimise} \text{ (resp. } \textit{maximise}) \ Z = q.\pi \\ &\textit{subject to the constraints } W\pi = p \\ &\qquad\qquad 1.\pi = 1, \ \pi_i \geq 0 \end{aligned}$$

where $p = \{p_1, \cdots, p_n\}$ is a n-vector, and q and π represent 2^n vectors of the assignment of c and probability distribution, respectively.

This method is applied in the formal proofs of the correctness of protocols, in which uncertain assumptions and postulates are permitted. BAN logic is given a probabilistic semantics, by which to express the insecure and uncertain environment. The belief in the conclusions depends on the probabilities assigned to the beliefs of principals and inference rules used in the proof. This study uses an example to describe how to calculate the probability in the authentication of server S of Needham-Schroedoer protocol in terms of the given assumptions and rules instances. It discovers the weakness of the protocol without using the prior knowledge. Although it is a promising method, it is not intuitive and too difficult to use.

Kailar proposes a special-purpose framework for the analysis of communication protocols that require accountability [82], such as the protocols for electronic commerce transaction. Unlike the current methods to reason about the evolution of beliefs of principals, this framework aims to reason about the ability of principals to prove accountability. It is not appropriate for the analysis of protocols, in which the accountability is not desired. Also, the message freshness or confidentiality is not considered.

2.6 Proof-Construction Approach

It is observed that inference-construction methods have limitations in handling secrecy. It is difficult to explain exactly what a belief-logic proof proves due to the lack of clear and complete semantics. Attack-construction methods may be subject to the exploration of search space in case of a complicated protocol. The proof-construction methods intend to deal with the above drawbacks.

Bolignano proposed an approach to verify authentication protocols [9]. It uses general purpose formal methods rather than modal logic to describe the protocol, hypotheses and authentication properties. The author claims that the use of general purpose formal methods can easily utilize the well-defined and largely used techniques.

This approach aims to obtain simple and concise proofs when a precise modelling of the protocol is required. In particular, it separates the modelling

of reliable agents and that of unreliable ones (intruders). Unlike Meadows's approach that focuses on proof simplification, it focuses on proof automation.

The transmission of a message is viewed as the combination of sending action and receiving action instead of an atomic action as the sending and reception of a message are not synchronous. To model the Needham-Schroeder protocol, they define 14 kinds of atomic actions in the form of $\mathcal{S} \times \mathcal{S}$, in which $\mathcal{S}$ represents the domain of possible state. The use of invariants and the axiomatization of intruder knowledge leads to a concise verification process as with conventional modal logic based methods. The first invariant property is that the private keys remain unknown to the intruder, and the second invariant is the fact that A and B use the correct public keys for the principal they want to talk to. The processes have been subjected to experiment in combination with the typed logics framework of the Coq prover [45].

An inductive method was presented by Paulson [129] to prove properties of security protocols. Unlike belief logics that permit short proofs to eliminate human error, inductive approaches include long and detailed proofs. The induction aims to prove every safety property over the protocol. The complicated proofs of this approach can be speeded up using the proof tool Isabelle/HOL, and a complete Isabelle proof script can be completed in a few minutes.

Traditionally, a protocol specifies a set of traces and is modelled using standard predicate calculus and set theory. Each trace may consist of a number of protocol runs. However, the recursive authentication protocol consists of arbitrary number of parties. For example, A contacts B. B may contact some other agents C or contact the authentication server. In either way, several fresh nonces or fresh session keys are generated. Such a protocol is hard to specify using traditional inductive methods.

Paulson presents a formal method to a variable-length protocol by recursive program, in which the number of steps, the number of participants and the number of session keys are not fixed [128]. It does not attempt to search for attacks, but to build guarantees for correctness properties. This method is partially automated by the Isabelle theorem prover.

2.7 Approaches Using Formal Tools and Specification Languages

The above methods can easily be applied to verify security protocols by the developers themselves. However, it is rather difficult for other analysts due to the fact that the protocols have to be redefined for each of the techniques developed and it is not easy to transfer the existing formal specification of the protocols into the used formal systems. Consequently, a number of formal specification languages and tools are developed for the purpose of automatic

protocol translation. They aim to design a unique protocol specification language, which can be used as the common input format for developed formal analysis techniques.

The infinite state space is a difficulty in verifying the security of cryptographic protocols. Furthermore, it is not easy to enable the automation of the language verification procedure in case of complicated protocols. Meadows proposes a heuristic approach for defining formal languages that can be automated in the most recent version of the Analyser [110]. In contrast, previous languages were manually specified. The approach is then used to prove the unreadability properties of languages automatically. This automatic language generator uses a simple format and speeds up proving that a word is or is not a member of a language, by which to improve the Analyser's performance. The rewrite rules of this approach use a standard format described as the form of $G \rightarrow X$, in which X represents a variable appearing in G. The features make it possible to prove unreadability results for other formal systems using rewrite rules.

A simple Interface Specification Language (ISL) is defined by Brackin. It is used to describe an Automatic Authentication Protocol Analyser(AAPA) which can automatically either prove that protocols satisfy the desired properties, or find where the proof attempts fail [16, 17]. The AAPA can be used either alone or as part of the *convince system*. The *convince tool* facilitates the modelling and analysis of cryptographic protocols using an automated support. The time and space required to do an AAPA analysis grow quadratically HOL theorem prover with the size of the protocol making it possible for the AAPA to analyse quickly large and complicated protocols. However, AAPA misses some failures, most notably non-disclosure failures, due to the fact that BGNY belief logic makes authentication deductions by assuming that there have been no non-disclosure violations.

Kemmerer presents an approach to analysing encryption protocols using machine-aided formal verification techniques [83] in terms of the formal specification language by Ina Jo [136]. Two specific symbols are used in this language.

N'' represents the new value of a variable,
T'' defines a subtype of a given type T

An example system is described by Kemmerer using the Ina Jo specification language. Ina Jo *criteria* clauses specify the critical requirements that the system is to satisfy in all states. Then theorems are generated to verify whether the requirements are satisfied or unsatisfied. Kemmerer detects a weakness in the sample system by this formal specification.

Sidhu [144] and Varadharajan [154] make use of state machine-based language to specify a protocol. A protocol is viewed as a collection of communication processes, one process for each entity. Each entity in the protocol

is represented as a finite state machine. A state transition has arisen due to the sending and receiving of messages. Three specific notations are used to describe the transition.

$-a$ represents the event of transmission of message a.
$+a$ represents the reception of message a.
$+\ a/-b$ represents the reception of message a followed by transmission of message b.

Varadharajan uses a state diagram for each entity to capture their actions in the protocol. Starting from the initial state, an arc is used to connect to the consequent state for each sent or received message. Figure 2.5 shows a simple state machine diagram for entity A. Let KDC be key distribution centre and let B be an arbitrary entity in the network. From the initial state S_0, A can

1. send the message a to KDC
2. receive the message c from another entity B who has acquired the session key from KDC.

If the (1) happens, the state S_1 is reached, and if (2) occurs, the state S_2 is reached. A can receive message c and message b, in the state S_1 and S_2, respectively. More states can be added to the state diagram for the entity A until all message transmissions of A are included.

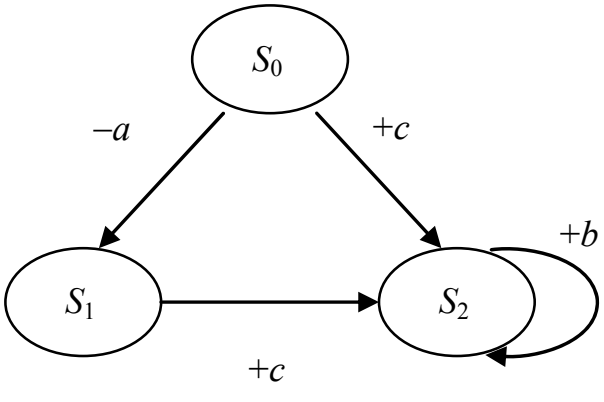

Fig. 2.5. State machine diagram for entity A

However, this method is complicated and not intuitive. In addition, it is assumed that A and B play the same role in the protocol. This collides with the assumption in [144], in which it is assumed that they have no symmetric role.

Another specification language, Common Authentication Protocol Specification language (CAPSL), is developed by Millen [115]. CAPSL is proposed as

a single common protocol specification language that can be used as the input format for any formal analysis techniques, such as Prolog state-search analysis tools [114]. Casper, developed by Lowe [99], semi-automatically produces the CSP description from a more abstract description, thus greatly simplifying the modelling and analysis process. Casper does not yet cover all the features found in security protocols, but has been applied to a number of known protocols, such as the Kerberos protocol [123], and Yahalom protocol [22].

It should be noted that although much research has concentrated on the attack-construction approach, proof-construction approach and formal specification language, most work in this field has been redirected because the inference-construction approach has been successful and attracted much attention due to its simplicity.

The verification model proposed in our book is based on ENDL. It inherits the properties of ENDL, which make it more suitable for the validation of electronic transaction protocols. Moreover, unlike the aforementioned methods, this verification model focuses on the formal analysis of secure electronic transaction protocols and is flexible enough to verify various security protocols by extension or adaption.

2.8 Summary

This Chapter aims to introduce some basic concepts and background knowledge. The primary security mechanisms to attain the security goal are classified into system security, network security, and data security. Among them, data and transaction security play an important role in e-commerce. The common way to achieve data security is encryption, and access control is often used to provide system security. In general, the security mechanisms are integrated into the security protocols to achieve security goals of different levels.

This chapter then describes the formalism, including the inference rules and basic notations. Three fundamental inference rules present the messages transmission and belief interrelationship.

A great many efforts have been put into the analysis of security protocols, including attack-construction methods, inference-construction methods and proof-construction methods. The aforementioned methods mainly employ theorem proving and state exploration tools, which are usually slow and complicated. BAN logic is a logic of belief. Its simple and intuitive inference rules make the formal analysis of protocol easy. Many other efforts are extended or adapted from this logic to different types of problems in cryptographic protocols. Recently, some attempts have been made to automate the verification of security protocols by model checking. On the other hand, a number of formal specification languages and tools are also developed to facilitate the description of protocols.

Although security protocols have been widely used for different security-related purposes, they are confronted with the challenge of subtle flaws due to the complexity of security protocols, diverse security properties, and a hostile environment. A security protocol uses cryptography to protect the transmitted data and to authenticate the other party. With the growth of computational power, the cryptography has evolved from the traditional substitution cipher and transition cipher to more complex public-key cryptography. However, a number of security protocols have been found to be subject to subtle flaws because designing them is a difficult and error-prone task. Several typical attacks are presented in this chapter.

Regardless of the varied cryptographic algorithms and methods of authentication, the formalization of security protocols tends to be similar. There have been considerable efforts to develop formal methods for security protocol analysis. Most of them are based on the Dolev-Yao model or BAN belief logic. However, they have shown limitations in detecting attacks owing to either exploration of state space or many ideal assumptions.

As more and more transactions such as financial transactions by e-commerce systems are performed electronically on open networks, and as more and more sensitive and confidential messages must be transmitted in some manner, concerns over information security challenge us and attract more and more attention. A number of security services including confidentiality, integrity, non-repudiation, authentication and authorization have been widely used to ensure reliable, trustworthy transmission of messages. Usually, they are achieved by using cryptography and are incorporated into security protocols.

Nevertheless, the design of security protocols is complicated and error-prone. Some of them have been found to contain subtle flaws. In the past twenty years, there have been considerable efforts to develop formal methods for protocol analysis. They have achieved good performance in most cases. Not only have plenty of special-purpose tools or adapted general-purpose tools been developed, but programmers have attempted to use them to create realistic protocols in many cases. The obtained feedback can be a good foundation for designers to improve the protocol's security.

3

Formal Analysis of Secure Transaction Protocols

3.1 Introduction

In today's information technology society, people are increasingly dependent on the internet for quality life. E-commerce systems have thus being presented attractive proliferation. The short list of facts below should be sufficient to place the current situation into perspective.

> *United States e-commerce transactions resulted in 707 million dollars in revenue in 1996, increasing to an impressive 2.6 billion dollars in 1997, and to an incredible high of 5.8 billion dollars in 1998. It is estimated that in 2007, nearly 40 million US households will book travel online, spending 86 billion on airline tickets, lodging, cars, intercity rail, cruises, and packages.* (*Forrester*).

As a result, a significant amount of research in the field of e-commerce has been carried out in the past decade, resulting in a variety of algorithms and techniques for electronic trading on the internet. A key problem for e-commerce users is how to carry out trade transactions on the internet safely. The security problems of e-commerce have mainly arisen from the fact that:

- data can be divulged: some important commercial information, including account number, is transmitted in plain text.
- information and source can be shared: a number of users in different locations can freely access other computers.

Security in e-commerce is implemented by relying on a set of security protocols that meet the user's expectation for secure business transactions. Communication between principals may be compromised without effective security protocols for key-exchange, authentication and privacy. Therefore security protocols, including cryptographic protocols and secure transaction protocols etc, have become the prime requisite of e-commerce systems. Ideally, these security policies should state the overall objectives of a transaction in terms

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 73–106, 2008.

of integrity, confidentiality and availability, and address the range of circumstances under which these objectives must be met.

Recently, there have been many remarkable efforts made on the security problem by developing methodologies, theories, logics, and other supporting tools. They are effective in overcoming the weakness and reducing the redundancies at the protocol design stage. However, there are also several instances of protocols advocated in [79, 120] which have been proved to be vulnerable to attack [65]. Therefore, it is so important to find ways to justify whether so-called secure protocols are secure. Without this justification, the users of electronic transactions will be on risk.

As already described, the existing approaches to validate security protocols can be mainly classified into three categories: *attack-construction approaches*, *inference-construction approaches* and *proof-construction approaches*. Although many subsequent methods are developed from them, as described in Chapter 2, they are subject to the exploration of search space, incomplete semantics or complexity. On the other hand, the features of electronic transaction cannot be handled by most of the traditional formal analysis of security protocols.

There have been attempts to analyse realistic e-commerce protocols, including the work of Kailar [82], Brackin [15], Meadows and Syverson [113] and Bolignano [10]. Recently, a number of approaches have been developed to analyse electronic commerce protocols from different aspects. An anonymous fair exchange e-commerce protocol that is claimed to satisfy fair exchange and customer's anonymity is analysed using OTS/CafeOBJ method [88]. In [127], a formalism that provides an expressive message passing semantics and sophisticated constructs for modelling principals is presented. A formal analysis based on Casper and FDR2 is proposed to analyse two electronic payment protocols: Visa 3D Secure and MasterCard Secure Code. It highlights issues concerning payment security in the proposed protocols. As to the electronic transaction using mobile devices, this complements protocol analysis techniques in terms of state enumeration. However, these security protocols are highly susceptible to subtle errors in real-world transactions. There are also many technical issues arising from cryptographic algorithms and network communication protocols [8, 41]. The limitations in existing validating approaches have been described in the former chapters.

As we have seen, it is very difficult to develop a verification approach that is suitable for all related protocols. In this chapter, a logical framework, ENDL [29], is proposed for validating secure transaction protocols [30]. The ENDL is referred as an inference-construction approach. It is developed since NDL contains some limitations, 1) it is not considerate on some aspects of practical financial circumstance, such as the certificate authentication; 2) it has been unable to accommodate some potential threats such as collusion attacks. The accumulation rule is further classified in terms of corresponding

situations, thereby it becomes more reasonable and applicable. By using the timestamp in validation, this can benefit the detection of replay attacks, which cannot be done by NDL.

Compared to traditional techniques, ENDL has some distinct features, such as *fail-negate*, *dynamic*, *non-monotonic*, *freshness*, *identity* and *random number*. Some properties are inherited from NDL. The timestamp has been used for modelling freshness in ENDL, which is the complement of BAN and GNY. These features enable us to solve the key problem of data integrity. To evaluate the logic, three practical instances of security protocols are illustrated. The results demonstrate that ENDL is effective and promising in analysing security protocols.

This chapter focuses on the design of a logical framework, ENDL. Several formal methods for analysing e-commerce protocols are presented in Section 3.2. Section 3.3 presents a computational model. In Section 3.4, a formal description of basic terms and statements of ENDL are provided. Section 3.5 establishes a basic inference framework and uses three practical transaction instances to evaluate ENDL. Section 3.6 concludes this chapter.

3.2 Research into Verifying Electronic Transaction Protocols

Electronic transaction operations depend on cryptographic protocols. However, weakness in encryption results in vulnerabilities. As a result, it is critical to develop techniques for comprehensive protocol analysis, especially to verify electronic transaction protocols due to the rapid growth of online trading.

In recent years, there have been various efforts to develop methods for the formal analysis of electronic transaction protocols. It may be difficult to introduce all of them in this book. Therefore, only representative methods are presented here.

3.2.1 Formalism for Protocol Analysis Using Process Calculi

A comprehensive analysis of protocols needs to model the complex message and knowledge and behaviour of principals. However, traditional logic techniques focus on modelling messages and the beliefs of principals, but do not provide mechanisms for expressing and reasoning about principal behaviour. Although process calculi capture the behaviour of principals, only limited facilities for modelling the knowledge and reasoning about it are provided. Therefore, Papa [127] presented a method for protocol analysis by combining logic and process calculus components.

One of the most important steps is to model the communication between principals as synchronous message passing. The initial definition includes the key, message, pattern and the message-pattern matching operator.

Modelling communication. A key is viewed as a public/private key, a secret key, the concatenation of two keys, or nil, for an unencrypted message. This can be expressed as:

$$key ::= K_n | K_n^{-1} | K_n^s | K_n : K_m | nil$$

It shows that a key is a public/private key (K_n/K_n^{-1}), a secret key (K_n^s), the concatenation of two keys ($K_n : K_m$), or *nil*. n and m in this formula represent the name of principals.

A message is specified as a list of values $\{v_1, v_2, \cdots, v_k\}$ encrypted by a *key*. A *value* may consist of a key, a message, a name or a fresh name $\sharp n$:

$$message ::= \{v_1, v_2, \cdots, v_k\}_{key}$$
$$value ::= key \mid message \mid n \mid \sharp n$$

Modelling principals. This presents an agent semantics that allows the comprehensive modelling of messages and principals. A system is viewed as zero or more concurrently executive agents. An agent is defined by its identifier *ID*, a list of values *lstv* (representing its knowledge), and a list of concurrent communication sequences *cseq*.

$$agent ::= ID[lstv](cseq)$$

In the similar way, a *concurrent sequence* (*cseq*), *annotated sequence* (*aseq*) and *sequence* (*seq*) can be defined by:

$$cseq ::= aseq \diamond cseq \mid nil$$
$$aseq ::= [seq].[lstv] \mid [seq]_\infty$$
$$seq ::= m^\frown seq$$

where $\diamond$ represents the concurrency operator for sequence and $[seq]_\infty$ represents an infinite sequence.

A concurrent sequence is the composition of an annotated sequence and a concurrent sequence. An annotated sequence is a sequence (*seq*) of message followed by a list of fresh names, or a sequence that has to be executed repeatedly. A sequence is specified as an output message m followed by a sequence, an exposed pattern followed by a sequence, or empty.

Suppose *C*, *M*, *S*, and *Prod* represent *customer*, *merchant*, *server* and *product*, respectively. NetBill protocol can be formally modeled as follows in terms of three steps, *negotiation*, *delivery* and *payment*.

1. $C \rightarrow M : \{Prod\}_{K_{CM}^S}$
2. $M \rightarrow C : \{Price\}_{K_{CM}^S}$
3. $C \rightarrow M : \{GoodsOrder\}_{K_{CM}^S}$
4. $M \rightarrow C : \{Goods\}_{K_G^S}$
5. $C \rightarrow M : \{\{PaymentOrder\}_{K_C^{-1}}\}_{K_{CM}^S}$

6. $M \rightarrow N : \{\{\{PaymentOrder\}_{K_C^{-1}}, K_G^S\}_{K_M^{-1}}\}_{K_{MN}^S}$
7. $N \rightarrow M : \{\{Result, K_G^S\}_{K_N^{-1}}\}_{K_{MN}^S}$
8. $M \rightarrow C : \{\{Result, K_G^S\}_{K_N^{-1}}\}_{K_{MN}^S}$

In the negotiation steps (1) and (2), the customer and merchant establish a price for the product. In the delivery steps (3) and (4), the customer places a goods order and the merchant delivers the encrypted product. In the payment steps (5), (6), (7) and (8), the customer pays for the order and obtains the decryption key.

In addition, this method presents three types of inference rules for specifying a formal semantics of agent behaviour.

- Agent communication rules. Three rules including **In**, **Out** and **Comm** are specified for agent communication. The **In** rule defines the behaviour of an agent that exposes a pattern and receives a message. The **Out** rule specifies the behaviour of an agent that outputs a message. Unlike the **In** and **Out** rules, the **Comm** rule defines reduction for a pair of communicating agents.
- Agent knowledge rules consist of **Knows**, **Extract** and **Construct** rules. The **Knows** rule represents predicates, in which the precondition shows that an agent knows a value v if v is in the agent's *list of values* or if the agent can be transformed into an equivalent agent that knows v. The **Extract** rule describes how to obtain values from a message in the *list of values* of an agent. The precondition requires the agent to have a valid key to access the message components. The **Construct** rule is used to generate new values from values known by an agent. The newly created values are concatenated with the agent's *list of values*. Nevertheless, the changes to the list of values only alter the representation of what the agent knows but do not create new agents.
- System development rules include **Chain** rule and **ChainBase** rule. **Chain** and **ChainBase** rules specify reductions for concurrent agents with multiple communication phases. The precondition of **Chain** is based on single reduction but the precondition of **ChainBase** uses agent equivalence.

The inference rules **In**, **Out**, **Comm**, **Chain** and **ChainBase** can be used to model agent behaviour and prove behavioural properties. Similarly, the rules **Knows**, **Extract** and **Construct** can be used to reason about the knowledge held by agents.

The technique is used to analyse NetBill protocol. The authors also make comparisons with state exploration, logic and process calculus based approaches. However, it is difficult to use due to complicated message passing semantics and concurrency constructs. And it is unclear whether the approach is well suited to verifying sophisticated electronic transaction protocols.

3.2.2 Formal Analysis Using an Observational Transition System

Fair exchange and anonymity are two important properties of electronic transactions. Some e-commerce protocols have claimed to satisfy the two requirements. To fully express them, not only the safety properties but also the liveness properties are needed. An approach [88] is proposed to formally analyse an e-commerce protocol, in which the protocol, together with the intruder, are modeled as an OTS (Observational Transition System). This transition system is written in *CafeOBJ*, which is an algebraic specification language.

The method presents how to model the protocol as an OTS; and how to write the OTS and express the safety part of the two requirements in *CafeOBJ*; and how to partially verify that the OTS satisfies the safety part by writing proofs or proof scores in *CafeOBJ*. It assumes that there exists a universal state space called Υ. An OTS S can be written as $<O, I, T>$.

- O represents a set of observers. Each $o \in O$ is a function $\Upsilon \rightarrow D$, where D is a data type. Let $v_1, v_2 \in \Upsilon$ be two states. The equivalence between two states $v_1 =_S v_2$ is defined as $\forall\ o \in O,\ o(v_1) = o(v_2)$.
- I represents a set of initial states and $I \subset \Upsilon$.
- T represents a set of conditional transition rules. Each $t \in T$ is a function $\Upsilon I =_S \rightarrow \Upsilon I =_S$. For each $v \in \Upsilon$, $t(v)$ be the successor state of v with respect to t.

An OTS S is described in CafeOBJ. An execution of S can be viewed as an infinite sequence $v_0, v_1, \cdots$ of states. It usually satisfies the two conditions below:

- *Initiation*: $v_0 \in I$.
- *Consecution*: $\forall\ i \in \{0, 1, \cdots\},\ v_{i+1} =_S t(v_i)$ for some $t \in T$.

The modelling of protocol consists of five phases in this approach.

1. **Assumptions**. It assumes that there is only one legitimate third party and there is only one legitimate bank for each customer and each merchant, respectively. The intruder can eavesdrop and glean any transmitted message in the network, however the intruder can decrypt a encrypted message or sign something only he/she knows the right key. Using the collected messages, the intruder can fake and send messages.
2. **Formalization of Messages**. There are 43 data types that are needed to formalize before formalizing messages. For example, **Customer**, **Merchant**, **CBank**, **MBank** and **Tparty** represent customers, merchants, customers' banks, merchants' banks and third party; constants **ic**, **im**, **icb**, **imb** and **itp** represent the intruder acting as a customer, a merchant, a customer's bank, a merchant's bank and a third party, respectively. The other data types can be found in [88]. After formalizing data types, messages can be formalized. Some operators are used to denote or construct

the messages. For example, **creatorm1**, **senderm1**, **receiverm1**, **s1m1** and **c1m1** will return the first argument, the second argument, the third argument, sig1 and cipher1 in message 1, respectively. *Sigi* and *cipheri* represent digital signature and cipher text, respectively.

3. **Formalization of the Network**. The network is modeled as a collection of messages. It is used as the intruder's storage to glean quantities and further fake messages. Also, the network is used as each principal's private memory that reminds the agent to send messages.
4. **Formalization of Trustable Principals**. The set of used random numbers and the networks are assumed to be observable. The observers are represented by CafeOBJ observation operators **ur** and **nw**, respectively. The operators are declared as follows:
   bop ur : System → URand
   bop nw : System → Network
   where **bop** denotes the declarations of observation and action operations start with bop. **Ur** and **nw** represent CafeOBJ observation operators. **Urand** is a set of random numbers that are used to generate really fresh random numbers.
5. Formalization of the Intruder. The intruder can fake messages based on the gleaned information. The intruder's faking messages are denoted by CafeOBJ action operators. Three action operators for faking message m_1 are listed below:

   bop fkm11 : System Customer Merchant Random Po Sig1 Cipher1 → System
   bop fkm12 : System Merchant Customer Po Random Random → System
   bop fkm13 : System Customer Merchant Random Price Random → System

According to the above assumptions and formalization, the verification phase is conducted to prove the required safety properties. Suppose a customer c conducts a transaction with a merchant m. It is assumed that c and m are trustable. The fair exchange property can be informally described below:

- If c received the ordered goods, then m has already been paid or will be ultimately paid for the goods.
- If m is paid for the goods ordered by c, then c has already received or will ultimately receive the goods.

In the similar way, the customer's anonymity can be informally described below:

- A customer uses a fake name to order goods from a merchant. The customer's bank account is never revealed.

The authors have formally or partly verified that the above safety properties holds for the protocol [74]. The verification was conducted by writing proof scores showing that the protocol satisfies these requirements in CafeOBJ and executing the proof scores with the CafeOBJ system.

3.2.3 Formal Analysis of Card-Based Payment Systems in Mobile Devices

Regardless of the rapid development of card based payment for online purchase of goods, the safety of electronic transactions has become a key issue to prevent its wider acceptance. The customers are concerned about the unauthorised spread of individual information to third party, theft of information kept by the merchants and the divulgence of credit card numbers, whereas the main concern of merchants and service providers is the authentication of card holders because the malicious user can use the stolen credit card number to make a purchase online.

To guarantee secure transactions, Visa and MasterCard independently proposed two electronic payment protocols: Visa 3D Secure [155] and MasterCard Secure Code [138, 139, 140]. The protocols aim to provide card holder authentication while they are conducting an electronic transaction using mobile devices. The protocols use pre-registered passwords to provide card holder authentication and Secure Socket Layer / Transport Layer Security (SSL/TLS) for data confidentiality over wired networks and wireless Transport Layer Security (WTLS) between a wireless device and a Wireless Application Protocol (WAP) gateway. A set of generic security goals applicable to electronic payment systems are proposed, and the formal tools Casper [57] and FDR2 [53] are used to analyse the protocols.

To participate in a payment system, both Visa and MasterCard require the card holders to enrol and register their passwords. Payment authentication is processed only if the card holder has previously enrolled with the right Issuer. The details with respect to enrollment and registration can be seen from the protocols but are ignored below.

Casper developed by Lowe is able to convert a high-level notation of the protocol to a Communicating Sequential Processes (CSP) script. This script can then be executed on a model checker, such as FDR, to verify whether the protocol achieves specific security objectives.

Firstly, each principal and intruder who can participate in a protocol are modeled as a CSP process. Casper is used to simplify the CSP description of protocols by allowing the user to specify the protocol at an abstract way. This script is then compiled in Casper and a CSP script is output, which is run by using the model checker FDR2.

Casper creates one refinement assertion for all secret specifications and one refinement assertion for each agreement and aliveness specification. A script

file includes statements making assertions about refinement features. Typically, the statements have the form as follows:

assert Abstract [X = Concrete
Example: Secret specification:
Secret(B, ban, [A])
Assertion generated:
SECRET_M::SECRET_SPEC[T=SECRET_M::SYSTEMS_S

The selected assertion is sent for testing by using FDR. The symbols associated with the assertion are updated to reflect the outcomes. The symbols projected by the FDR are:

- Tick ($\surd$) shows that the test is completed successfully.
- Cross (X) shows that the test is completed but the refinement does not hold. The FDR can then be used to find out the reason for the failure.
- Exclamation mark (!) shows that the test failed to complete due to: a syntax or type error in the scripts, exhausting resource while trying to run the test, or interrupted test.
- Zig-zag (Z) shows that FDR cannot finish a test owing to a fault in coded algorithms.

If a refinement is found to be unsatisfied, then the protocol might contain a weakness. The weakness in the protocol is checked by observing the trace leading to divergence.

An important process to model the protocols is to reserve important protocol mechanisms while simplifying the protocols. Casper representation of the protocols includes the SSL representation and certificates. The protocols use SSL (Secure Socket Layer) to provide security for data transmission. As to protocol analysis using Casper/FDR, it assumes:

1. The underlying cryptographic algorithms of public key and symmetric key are robust.
2. The client and server successfully negotiated the SSL to generate a symmetric session key.

In addition, this assumes that all agents unconditionally trust the certification authority and public key signed by it. Nevertheless, the distribution of certificates is ignored and all certificates held by the protocol participants have been validated by the certification authority.

Visa and MasterCard do not clearly specify any security goals, whereas a generic set of security goals are proposed to verify the protocol. The goals are classified into four categories, including data security, payer security, payee security and transaction security. The formal analysis of the protocols are presented in terms of each security goal.

Data security represents that the third party that does not involve in the payment system in an electronic payment system should not gain access to the participants' transactional data or their secret keys. The data security can be described in terms of *third party*, *privacy* and *SSL keys*.

Third party. For Visa 3-D Secure, the security requirement can consist of C's values *pan* (primary account number) and *edate* (expiry date), which should be known to ACS (access server), M and DS (domain server) only. The process can be represented in an abstract way:

StrongSecret(C, *pan* [ACS, M, DS])
StrongSecret(C, *edate*, [ACS, M, DS])

As to MasterCard Secure Code, it requires that the C's values *pan* and *expiry* should be known to ACS and M only.

StrongSecret(C, *pan* [ACS, M])
StrongSecret(C, *expiry*, [ACS, M])

Privacy represents that the payer's payment information should not be known by the payee and the payer's order information should not be known by the bank. An example of Casper specification with respect to C's password of Visa 3D Secure is represented as:

StrongSecret(C, *password*, [ACS])

In the similar way, *SSL keys* can be defined. Both protocols use a shared SSL session key to encrypt messages transmitted between participants. For example, a session key shared between M and C and the verification of its agreement can be represented in Casper specifications:

StrongSecret(C, *keyMC*, [M])
StrongSecret(M, *keyMC*, [C])
Agreement(C, M, *keyMC*)
Agreement(M, C, *keyMC*)

Payer security is described from two aspects, authentication, and authorization and acceptance. The former represents that the payer should have proof of other agent's authenticity before communicating with that agent. Nevertheless, the latter means that the payer should have proof of transaction authorization by the bank and transaction acceptance by the payee for a transaction.

Payee security. Similar to the payer security, the payee security consists of authentication and authorization, but no acceptance. Thus, the payee should

obtain proof of other agent's authenticity before transactions, transaction authorization from the payer, and transaction authorization from the bank.

Bank security includes authentication, authorization and processing request. Authentication of card holders and vendors is via password. The Casper representation for proving the authentication of card holder via the bank and directory server is represented as:

Agreement(C, ACS, [*password*])
Agreement(ACS, C, [*password*])
StrongSecret(ACS, *password*, [C])

The bank needs to obtain an authorization proof for transactions from the card holder according to the successful authentication of card holder and proof of agreement on amount and merchant. For example, the Casper representation for proving authorization of payment amount and current payment date-time by the card holder is described as:

Agreement(C, ACS, [*pamt*])
Agreement(C, ACS, [*pdt*])

On the other hand, the bank should obtain proof of transaction processing request from the merchant by validating the payment authorization request from the merchant.

Transaction security. Every transaction process should be unique. In Visa 3D secure, the uniqueness is obtained by using the fresh generation of transaction *id* by the merchant, and its verification by the bank. In MasterCard Secure Code, transaction uniqueness is achieved by using the fresh transaction number generated by the merchant.

In this method, SSL is incorporated into the protocol representation since Visa 3D secure and Master Card secure depend on SSL to provide confidentiality for messages. An attack on Visa was detected from the protocol, which is based on the merchant being dishonest. Although most customers trust known merchants, the smaller merchants may lack the trustworthiness in the future in contrast to more well established merchants.

3.3 A Computational Model

Figure 3.1 depicts a computational model for e-commerce. There are two parts in the figure: a public environment and a secure transaction protocol. The public environment is an *open network* that is composed of principals connected by communication links. All kinds of messages on these links, such

as identity, account number and key, constitute the communication between principals. The principal not only places messages on a link, but also sees or modifies the messages on a link. An open network is considered as a set of principals (users, hosts, and processes) and messages. Principals can interact with each other according to the rules of some predefined protocols in order to accomplish a common task (e.g., to encrypt a message). Interactions are based on messages that are conveyed via a communication facility.

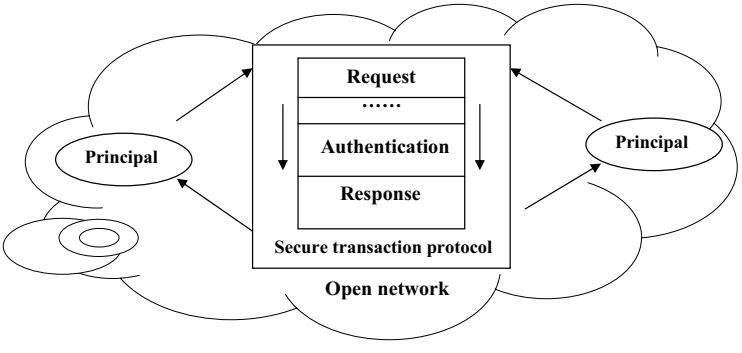

Fig. 3.1. A computational model

The *secure transaction protocol* is a method used to secure transactions over an open network and is carried out by the principals. The protocol is organised into several stages by message transmissions, such as a request for certificating authority. In these stages, cryptography is used to provide confidentiality of information to ensure data integrity and to authenticate the participants. A special execution of protocol is called a 'session'. For convenience, this chapter only considers sessions that appear to end successfully, and the overhead issues are excluded.

PKI (public key infrastructure) tree: As principals receive a certificate, they need to verify the validity of the certificate through a hierarchy of trust. Each certificate is linked to the signature certificate of the entity that digitally signed it. By following the trust tree to a known trusted third party, namely *CARoot*, one can be assured that the certificate is valid if it is able to pass the rigorous verification of the certificate authority at each level of the PKI tree. This has been proved to be efficient in [140]. Figure 3.2 illustrates the hierarchy of trust.

All certificates must be verified through this structure. *CARoot* is the topmost level of certificate authority, and every principal should trust its reliability. For instance, a customer certificate is linked to the certificate of the issuer. The issuer's certificate is linked back to a *CARoot* key through the Brand's certificate. Because all participants know the public signature key of

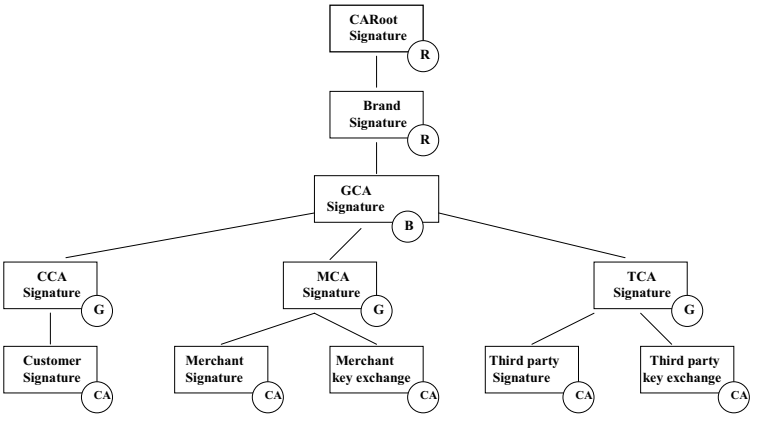

Fig. 3.2. PKI tree of Trust

the *CARoot*, it may be used to verify each of the certificates in turn. The number of levels shown in this diagram is illustrative only. In practical transactions, a customer may not always verify the certificate through complete certificate authority. Thus we use the chain of trust, CA=$\{CA_1, CA_2, \ldots, CARoot\}$, to express the related certificate authority in a general way. We do not care which authority appears in the chain of trust. The initial level in the certificate indicates which Certificate Authority created the certificates, and GCA indicates the Brand Geo-political CA.

At each stage of a protocol session, every principal has a finite set of *received* data items that it had before the session began, or that were extracted from the messages sent to it before, or during the current stage. However the potential set of information is infinite.

Also, if principal A authenticates the messages m sent from principal B, then A *believes* m. This indicates that principal A has enough confidence in message m. Although this confidence may not be very high, A still believes m is true.

After receiving a message, the principal starts verifying the message using inference rules and related axioms. The messages and verification form the initial set of received data items and their confidences for processing next time. During the verification, if the principal cannot conclude that 'principal B knows message m', then the principal believes 'B does not know m'.

To explore a flaw-free protocol, a logic, written as NDL (non-monotonic dynamic logic), has been established in [6]. The NDL is a natural and practical analysis strategy for the verification of security protocols. It has well-integrated techniques taken from single-rank logic, dynamic logic and non-monotonic logic, for efficiently identifying flaws. However, the NDL has been unable to accommodate the actual circumstances of secure transaction pro-

tocols such as data integrity. A logical framework ENDL is then developed to deal with the above issues. For simplicity, it assumes that all principals involved in a transaction do not divulge their secret, especially the root of the certificate issuer. Also, it assumes that the cryptographic algorithms and communication protocols are sound.

3.4 Basic Terms and Statements

This section presents formal definitions for the function word, predicate and action, which are used for building our ENDL logic as follows:

Principal is some participant who carries out the protocol.
Uppercase X, Y, A, B, C, and CA (Certificate Authorities) range over particular principals.
m_1, m_2, ..., and m_n denote specific messages. (Here, keys and the encrypted messages are viewed as messages either.)
T denotes a specific timestamp that can be used to authenticate the validity of message and to assert that the message is created for a current session.
Cert denotes the certificate that needs to be verified.
CertReq denotes the registration form of the request for certificate.
k denotes the encryption, or decryption, keys.
Generate and *Send* denote specific actions. (Encryption and digital signature etc. are some mapping operations on messages, but not actions.)
P and Q denote formulae, on which the infrastructure of ENDL can be constructed by employing the following components:

Knowledge state is applied to denote a specific knowledge relationship between principals and messages within the period of protocol execution. For instance, *Alice* knows message m sent by *Bob* implies two kinds of relationships: one is the recipient *Alice* receives and knows m, and the other is the initiator *Bob* sent m.

Fresh is applied to a process and asserts this term was created for a current session but not recorded from an earlier session. A principal believes a term is fresh if it can identify the term as created for the current session whenever this term comes into its possession. To consider the possibility that communication keys may be compromised in an open network and the timestamp may prevent replays, timestamp T is introduced for ensuring freshness of message. If the clock synchronization of both parties is difficult, a trusted third party can intervene as a notary and uses its own clock as a reference [87, 143]. Actually, the timestamp has the additional benefit of replacing the two-step handshake in the Needham and Schroeder protocol for a public key system, but the exposure of a user's private keys cannot be eliminated by the timestamp.

Unless compromises are reported to the AS (authentication server) and public keys are obtained from the AS immediately prior to use, timestamps can be useful for validating the time integrity of keys [40].

Identification is the process of ascertaining the identity of a participant by relying on uniquely distinguishable features, known as identifier *ID*. This contrasts with authentication, which is the confirmation that the distinctive identifier indeed corresponds to the declared user. The recipient proposes privately the verifier a secret that is shared only with the sender, of which authentication and identification take place simultaneously, such as a password or a secret encryption key.

Authentication denotes specific belief relations. The purpose of authentication of participants is to reduce, if not eliminate, the risk that intruders might masquerade under legitimate appearances to pursue unauthorized operations.

Integrity is applied to guarantee that messages come from the purported sender, and that the message content is not alerted during transmission between the originator and the recipient. Both *integrity* and *authentication* can be ensured by using digital signature.

Random number: Detection of duplication and replay is achieved with the use of a random number, namely *Rnd*, before encryption.

Confidentiality: Applied to guarantee that information is safe and can only be accessed by the intended recipients. It is realized by encryption.

Function word: The abstract description of the operations on a message. They consist of encryption, signature, message digest, and associated mapping relation of the key.

$e(m, k)$: This represents the operation that message m is encrypted by the symmetric (communication) key k.
$E(m, k)$: This represents the operation that message m is encrypted by the public key k, namely $Kpb(X)$ and $Spb(X)$ listed below.
$S(m, k)$: This represents the operation that message m is encrypted by the private key k, namely $Kpv(X)$ and $Spv(X)$ listed below.
$H(m)$: This represents the message digest of message m encoded by the one-way hashing algorithm $H(x)$. The one-way Hash function has the property that, given the output, it is difficult to determine the input.
$Kpb(X)$: This represents the public key-exchange key of X.
$Kpv(X)$: This represents the private key-exchange key of X.
$Spb(X)$: This represents the public signature key of X.
$Spv(X)$: This represents the private signature key of X.
$<m_1, \ldots, m_n>$: This represents the combination of messages $m_1, \ldots,$ and m_n.

Although $Kpv(X)$ and $Kpb(X)$ are restricted to exchanging asymmetric keys in this chapter, they actually can be used for confidential exchange of any message.

When the message digest of a message is encrypted using a sender's private key, and is appended to the original message, the result is known as the digital signature of the message (see abbreviation below.) The function word is the infrastructure necessary to describe the complicated cryptographic operations. Moreover, the related cryptographic algorithms used to construct the encryption and decryption are basically believed to be robust, but we cannot absolutely exclude the possibility that some intelligent attackers can intercept the message and break down the encryption successfully within, or before, the expiry date. For simplicity, this chapter primely demonstrates how to apply our approach to verify security protocols.

Action is applied to describe the communication process in which a principal is the executant of the action and tries to execute some appointed tasks. There are two types of actions below:

> $Generate(X, m)$: Applied to represent that X generates the message m.
> $Send(X, Y, m)$: Applied to represent that X sends the message m to Y after X generated the message m successfully.

Applying the conventional logic operator can derive further actions. Suppose α and β are basic action sequences, then $\alpha\ o\ \beta$ denotes the conjunction of α and β, and can be treated as an action sequences (see Inference Rule below). According to the requirements of analysis of the practical secure transaction protocols, we are able to add new actions.

Predicate is applied to express the knowledge state and belief relation of principals. There are four kinds of predicate:

> $Know(X, m)$: This represents that X knows message m. It is possible that X generates the message m or receives m from Y. However, some malicious attacks can make X forget message m even within its period of validity, which will be described later.
> $Auth(X, Y, m)$: This represents that X authenticates message m sent by Y and m has not been modified. If X can authenticate the message m is valid, then the return value is true; otherwise the return value is false.
> $IsVerified(X, Cert)$: This represents that, if X can verify the $Cert$ is valid, then it returns true, otherwise it returns false.
> $Equal(m, m')$: This represents that, if message m is equal to message m', then it returns true, otherwise it returns false.

Assertion

$$\boldsymbol{P} \vdash_{\alpha} \boldsymbol{Q}$$

P and Q present a set of formulae; and α denotes a series of actions. This assertion denotes that if the premise P is true then α can be executed, and the conclusion Q will be true if α can be performed successfully. Let P_1 and P_2 be formulae, then the followings are also regarded as formulae:

$$P_1 \wedge P_2 : P_1 \text{ and } P_2$$
$$P_1 \vee P_2 : P_1 \textit{ or } P_2$$
$$P_1 \longrightarrow P_2 : P_1 \text{ implies } P_2$$

'$\wedge$', '$\vee$', and '$\longrightarrow$' are some traditional logical operators. We also imply some notations from set theory. Suppose we have a formula $P = P_1 \wedge P_2$. We can say $P \in P_1$ and $P \in P_2$, which means P is the intersection of P_1 and P_2. Similarly, suppose the formula is $P = P_1 \vee P_2$, we can say $P \in P_2$ or $P \in P_2$, which means P is the union of P_1 and P_2.

Abbreviation

We define four expressions, abbreviated to simplify the conjunctions of several function words, since these conjunctions repeatedly appear in this logical system. The following abbreviations, in fact, imply complex cryptographic operations, digital signature, message digest and the like.

$Sign(X,\ m)$=<m, S(<ID_X, $H(m)$>, $Spv(X)$)>: This means that plain text m and X's identifier ID_X are attached to X's digital signature.
$Sign(X, m)_T$= <m, S(<ID_X, T, $H(m)$>, $Spv(X)$)>: Inserting the timestamp T into $Sign(X,\ m)$.
$S_0(X,\ m)$= S(<ID_X, $H(m)$>, $Spv(X)$): This means that identifier ID_X was attached to X's digital signature before X encrypted the message digest of m in the private signature key $Spv(X)$.
$S_0(X, m)_T$ = S(<ID_X, T, $H(m)$>, $Spv(X)$): Inserting the timestamp T into $S_0(X,\ m)$.
$CertK(X) = Sign(CA, <X, Kpb(X)>)$: This means that the key-exchange certificate of X.
$CertS(X) = Sign(CA, <X, Spb(X)>)$: This means that the signature certificate of X.
$Verify(X, Cert, <CA_1, \ldots, CARoot>)$: Here X verifies certificate $Cert$ by traversing the trust chain, $CA_1, CA_2, \ldots$, to the root $CARoot$(see above under **PKI** tree).

3.5 Logical Framework and Statement of ENDL

This section proposes the ENDL framework, including its axioms and inference rules; and the inference format comprising the accumulation property.

3.5.1 Axiom

(1) Encryption

1-1 $Know(X, m) \wedge Know(X, k) \longrightarrow Know(X, e(m, k))$

This means that, if X knows message m and communication key k, then X knows $e(m, k)$ by using k to encrypt message m. It is unnecessary for X to be concerned with the reliability of message m and key k. Since X knows m and k, there is no doubt that X can know $e(m, k)$. Another kind of encryption, namely public key-exchange key encryption, can be expressed by the following formula:

1-2 $Know(X, m) \wedge Know(X, Kpb(Y)) \longrightarrow Know(X, E(m, Kpb(Y)))$

This means that, if X knows message m and public key-exchange key $Kpb(Y)$ of Y, then X knows $E(m, Kpb(Y))$ by using $Kpb(Y)$ to encrypt message m.

1-3 $Know(X, m) \wedge Know(X, Spv(Y)) \longrightarrow Know(X, S(m, Spv(Y)))$

This means that, if X knows message m and private signature key $Spv(Y)$ of Y, then X knows $S(m, Spv(Y))$ by using $Spv(Y)$ to encrypt message m.

In a practical transaction, protocol could be insecure when user's keys are compromised. To solve this issue, a method using timestamp T has been proposed by Denning [40]. Along with other information the messages that need to be protected are appended with a timestamp before encryption. Also, Gong has proposed a security risk of depending on synchronized clocks [61]. According to Gong, clocks can become unsynchronised due to sabotage on, or faults in, the clocks themselves or the synchronization mechanism. Thus, security can be jeopardised by overflows and the dependence on potentially unreliable clocks on remote sites. Despite this, this framework still uses the assumption of a network of synchronized clocks. Therefore, it assumes that a faulty clock can be resynchronised efficiently.

(2) Key Allocation

2-1 $Know(X, Kpb(CARoot))$

That is all principals of security protocol know the public key-exchange key of *CARoot*.

2-2 $Know(X, Spb(CARoot))$

That is all principals of security protocol know the public signature key of *CARoot*.

2-3 $Know(X, Kpv(X))$

That is X knows its own private key-exchange key. It is computationally unfeasible for anybody to deduce it from the public key-exchange key.

2-4 $Know(X, Kpb(X))$

That is X must know its own public key-exchange key.

2-5 $Know(X, Spv(X))$
That is X knows its own private signature key. It is computationally unfeasible for anybody to deduce it from the public signature key.

2-6 $Know(X, Spb(X))$
That is X must know its own public signature key.

(3) Decryption

3-1 $Know(X, k) \wedge Know(X, e(m, k)) \longrightarrow Know(X, m)$
That is, if X knows communication key k and $e(m, k)$, then X knows the plain text of message m by using k to decrypt $e(m, k)$.

3-2 $Know(X, Kpv(Y)) \wedge Know(X, E(m, Kpb(Y))) \longrightarrow Know(X, m)$
That is, if X knows private key-exchange key $Kpv(Y)$ and $E(m, Kpb(Y))$, then X knows the plain text of message m by using the private key-exchange key $Kpv(Y)$ to decrypt $E(m, Kpb(Y))$.

3-3 $Know(X, Spb(Y)) \wedge Know(X, S(m, Spv(Y))) \longrightarrow Know(X, m)$
That is, if X knows the public signature key $Spb(Y)$ of Y and $S(m, Spv(Y))$, then X knows the plain text of message m by using the public signature key $Spb(Y)$ of Y to decrypt $S(m, Spv(Y))$.

Contrasting with the encryption operation, decryption is a reverse process. It may be worth examining whether the message, if transmitted in an open network, can suffer from a malicious attack (e.g. masquerading, replay, and eavesdropping) when a user is receiving an encrypted message. A user must decrypt it, and then try to verify whether the result can be trusted or not. All these processes must conform strictly to the requirements of the security protocol.

(4) Signature

4-1 $Know(X, m) \longrightarrow Know(X, H(m))$
That is, if X knows m, then X knows $H(m)$.

4-2 $Know(X, m) \wedge Know(X, Spv(Y)) \longrightarrow Know(X, S(H(m), Spv(Y)))$
That is, if X knows message m and private signature key $Spv(Y)$ of Y, then X knows the message digest $H(m)$ (derived from axiom 4-1) and $S(H(m), Spv(Y))$ by using $Spv(Y)$ to encrypt $H(m)$ (derived from axiom 1-3).

Exposure of private keys in public key systems poses a serious threat. If a user's private key is exposed, all messages encrypted by the corresponding public key may be compromised. In addition, an attacker will also be able to sign messages on behalf of the user. There are two possibilities: exposure of user's private keys and exposure of the private key used by the certificate authority to sign certificates. Section 3.5.4 will show that this logic uncovers some known flaws in security protocols.

(5) Authentication

5-1 $Know(X,\ m) \wedge Know(X,\ S(<ID_Y,\ T,\ H(m)>,\ Spv(Y))) \wedge Know(X,\ Spb(Y)) \xrightarrow{|Clock-T|<\triangle t_1+\triangle t_2} Auth(X,\ Y,\ m)$

This means that, if X knows message m, $S(<ID_Y,\ T,\ H(m)>,\ Spv(Y))$ and $Spb(Y)$, then X can authenticate that Y sent m, and m has not been modified by using $Spb(Y)$ to verify $S(<ID_Y,\ T,\ H(m)>,\ Spv(Y))$ (derived from axiom 3-3) and then comparing the result with a newly generated message digest of m. Simultaneously, X verifies that a message is not replayed by checking that $|Clock\ -T|<\triangle t_1+\triangle t_2$, where $Clock$ is the local time, $\triangle t_1$ is an interval representing the normal discrepancy between the server's clock and the local clock, and $\triangle t_2$ is an interval representing the expected network delay time [40].

5-2 $Know(X,\ m) \wedge Auth(X,\ Y,\ H(m)) \longrightarrow Auth(X,\ Y,\ m)$

This means that, if X knows message m and authenticates that Y actually sent $H(m)$, and that it has not been modified, then X can authenticate that Y actually sent m, and m has not been modified. In this formula, it is unnecessary to add the predicate for expressing that X knows the public signature key $Spb(Y)$ of Y , since X is convinced of the validity of $H(m)$.

5-3 $Know(X,\ Spb(Y)) \wedge Know(X,\ S(<Y,\ T,\ Spb(Y)>,\ Spv(CA))) \wedge Know(X,\ Spb(CA)) \xrightarrow{|Clock-T|<\triangle t_1+\triangle t_2} Auth(X,\ Y,\ Spb(Y))$

That is, if X knows $Spb(Y)$, and $S(<Y,\ T,\ Spb(Y)>,\ Spv(CA))$, then X can authenticate $Spb(Y)$ by using the public signature key of CA to verify the encrypted message and checking the timestamp and identity included in the message.

(6) Separation

6-1 $Know(X, <m_1, \ldots, m_n>) \longleftrightarrow Know(X,\ m_1) \wedge \ldots \wedge Know(X,\ m_n)$

This means that, if X knows a compound message $<m_1,\ m_2,\ \ldots\ldots,\ m_n>$, then X should know every component of it. The reverse is also true, since X knows each of them, X can combine them to form a new group of messages.

6-2 $Auth(X,\ Y,\ <m_1,\ \ldots,\ m_n>) \longrightarrow Auth(X,\ Y,\ m_1) \wedge \ldots \wedge Auth(X,\ Y,\ m_n)$

That is, if X authenticates that Y ever sent a compound message $<m_1,\ m_2,\ \ldots\ldots,\ m_n>$ that has not been modified, then X authenticates that Y ever sent every element of it and none of it has been modified. The reverse is not true since X may authenticate each message at a different time but the belief on them can change as time passes, thus X cannot authenticate the compound message $<m_1,\ m_2,\ \ldots\ldots,\ m_n>$ even though it could authenticate its every component beforehand.

(7)PKI

7-1 $Know(X,\ CertS(Y)) \wedge Verify(X,\ CertS(Y),\ CA) \wedge IsVerified(X,\ CertS(Y)) \longrightarrow Auth(X,\ CA,\ <Spb(Y),\ Spv(Y)>)$
This means that, if the signature certificate of Y, $CertS(Y)$, issued by CA (certificate authority) is verified successfully by X, then X can authenticate that the public signature key $Spb(Y)$ of Y is valid and issued by CA.

7-2 $Know(X,\ CertK(Y)) \wedge Verify(X,\ CertK(Y),\ CA) \wedge IsVerified(X,\ CertK(Y)) \longrightarrow Auth(X,\ CA,\ <Kpb(Y),\ Kpv(Y)>)$
This means that, if the key-exchange certificate $CertK(Y)$ of Y issued by CA is verified successfully by X, then X can authenticate that the public key-exchange key $Kpb(Y)$ of Y is valid and issued by CA.

These two axioms are used to describe how a principal verifies the CA certificate by traversing the hierarchy of the trust chain to the *CARoot*. Once the certificate is authenticated, the user will hold a copy of the certificate to use later.

Below, two theorems for the logic are described. They will be used in the construction of the next lot of inference rules.

Theorem 3.1. *If a principal X knows a message m, and the public signature key of Y and Y's digital signature on m which includes timestamp T, then X may authenticate that the message m is sent by Y and has not been modified.*

$$Know(X,\ m) \wedge Know(X,\ Spb(Y)) \wedge Know(X,\ S_0(Y,\ m)_T)) \xrightarrow{|Clock-T|< \atop \triangle t_1+\triangle t_2} Auth(X,\ Y,\ m)$$

[Proof]:

(1) $Know(X,\ m)$ [premise]
(2) $Know(X,\ S_0(Y,\ m)_T)$ [premise]
(3) $Know(X,\ Spb(Y))$ [premise]
(4) $Know(X,\ S(<ID_Y,\ T,\ H(m)>,\ Spv(Y)))$ (2)[definition]
(5) $Auth(X,\ Y,\ m)$ (1)(3)(4)[5-1]
(6) $Know(X,\ m) \wedge Know(X,\ S_0(Y, m)_T) \wedge Know(X,\ Spb(Y)) \longrightarrow Auth(X,\ Y,\ m)$ (1)(2)(3)(5)[$\rightarrow +$]

The proof can easily be constructed by using the authentication axiom 5-1 and the definition of $S_0(Y, m)_T$, which is an equivalent expression of $S(<ID_Y,\ T,\ H(m)>,\ Spv(Y))$.

Theorem 3.2. *If a principal X knows the public signature key of Y and Y's digital signature on m, which includes timestamp T, then X can authenticate that the message m is sent by Y and has not been modified.*

$$Know(X,\ Spb(Y)) \wedge Know(X,\ Sign(Y,\ m)_T) \xrightarrow{|Clock-T|< \atop \triangle t_1+\triangle t_2} Auth(X,\ Y,\ m)$$

[Proof]:

(1) $Know(X, Sign(Y, m)_T)$ [premise]
(2) $Know(X, Spb(Y))$ [premise]
(3) $Know(X, <m, S(<ID_Y, T, H(m)>, Spv(Y))>)$ (1)[definition]
(4) $Know(X, m)$ (3)[6-1]
(5) $Know(X, S(<ID_Y, T, H(m)>, Spv(Y)))$ (3)[6-1]
(6) $Know(X, S_0(Y, m)_T)$ (5)[definition]
(7) $Auth(X, Y, m)$ (2)(4)(6)[Theorem 3.1]
(8) $Know(X, Sign(Y, m)_T) \wedge Know(X, Spb(Y)) \longrightarrow Auth(X, Y, m)$ (1)(2)(7)[$\rightarrow +$]

This proof of Theorem 3.2 is directly derived from the separation axiom 6-1, the definition of $Sign(Y, m)_T$, which is an equivalent expression of $<m, S(<ID_Y, T, H(m)>, Spv(Y))$, and the definition of Theorem 3.1.

The above two simple theorems are common processes in security protocols and provide a compendious and perspicuous way to describe the operation of digital signature.

3.5.2 Inference Rules

As already described, the inference rules of NDL consists of (R-1) *Revelation*, (R-2) *Generation*, (R-3) *Accumulation*, (R-4) *Union*, and (R-5) *Nonmonotonic*. In ENDL logic, all the inference rules except for *Accumulation* keep consistent with NDL. However, it sorts the original accumulation rule into several categories in light of the potential message lost and attacks.

In our former work [27], it assumed that the key cannot be altered and the principals have the property of memory to message m, but the key, in fact, can be altered and the principal may not possess memory to m. Thus, the accumulation rule (R-3) is classified according to the circumstance that can possibly happen in a practical e-commerce environment.

(1) Some of k, $<Spb(X), Spv(X)>$ and $<Kpb(X), Kpv(X)>$ are altered, but the principal has memory to message m.
(2) None of k, $<Spb(X), Spv(X)>$ and $<Kpb(X), Kpv(X)>$ are altered, but the principal has no memory to message m.
(3) Some of k, $<Spb(X), Spv(X)>$ and $<Kpb(X), Kpv(X)>$ are altered, and the principal has no memory to message m.
(4) None of k, $<Spb(X), Spv(X)>$ and $<Kpb(X), Kpv(X)>$ are altered, and the principal has memory to message m.

If all the keys are not altered and the principal has memory to message m, thus we may still apply the accumulation rule (R-3) and do not need to be concerned about the possibilities described above.

In circumstance (1), principal X randomly alerts cryptographic keys, k or $<Spb(X),\ Spv(X)>$ or $<Kpb(X),\ Kpv(X)>$, and lets Y know the exact key after alteration. The accumulation rule (R-3) can still be applied since the principals X and Y, have memory to message m, which helps them to recall message m. However, if X alters the cryptographic keys, k or $<Spb(X),\ Spv(X)>$ or $<Kpb(X),\ Kpv(X)>$, and does not let Y know the correct key promptly, then the knowledge of Y on the former cryptographic keys is invalid. Thereby, the accumulation rule (R-3) cannot be used again.

Based on the above two particular circumstances, four corresponding rules can be deduced as shown below.

(R-I-1) Accumulation 1

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ \ newkey),\ Know(Y,\ \ newkey)}}{\boldsymbol{P \vdash_{\alpha\ o\ Generate(X,\ newkey)} Q}}$$

That is, if X alters the former cryptographic keys, generates *newkey*, and lets the corresponding principal know them, then the conclusion Q that has been proved to be true remains correct after the action $\alpha\ o\ Generate(X,$ *newkey*$)$, which consists of *Generate*() and *Send*(), and so forth. In this rule, *newkey* may only be any combination of k, $Spb(X)$ and $Kpb(X)$ because the private signature key $Spv(X)$ and private key-exchange key $Kpv(X)$ cannot be disclosed to Y since they contain personal privacy that must be kept secret to other principals.

In circumstance (3), if only X does not alter the message, this rule can actually be applied to this special instance.

(R-I-2) Accumulation 2

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ \ <newkey,\ m>),\ Know(Y,\ \ newkey),\ Equal(m,\ \ m')}}{\boldsymbol{P \vdash_{\alpha\ o\ Generate\ (X,\ <newkey,\ m>)} Q}}$$

That is, if X changes the former cryptographic keys, k or $<Spb(X),\ Spv(X)>$ or $<Kpb(X),\ Kpv(X)>$, and message m', and lets Y know the *newkey*, then, if m is equal to m', the conclusion Q that has been proved to be true keeps correct after the action $\alpha\ o\ Generate(X,<$*newkey*, $m>)$. The action consists of *Generate*() and *Send*() and so forth. Because the principals have memory for message m' so Y can easily compare m' with m. In this rule, *newkey* may only be any combination of k, $Spb(X)$, $Kpb(X)$ because the private signature key $Spv(X)$ and private key-exchange key $Kpv(X)$ cannot be disclosed to Y since they have personal privacy that must be keep secret from other principals.

(R-I-3) Accumulation 3

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ \ <newkey,\ m>),\ Know(Y,\ \ newkey),\ \neg Equal(m,\ \ m')}}{\boldsymbol{P \mid\sim_{\alpha\ o\ Generate\ (X,\ <newkey,\ m>)} \neg Q}}$$

That is, if X changes the former cryptographic keys k or $<Spb(X), Spv(X)>$ or $<Kpb(X), Kpv(X)>$, and message m', and lets Y know the *newkey*, then if m is not equal to m', we can say non-monotonically that conclusion Q that has been proved to be true is incorrect after the action α *o* $Generate(X, <newkey, m>)$.

(R-I-4) Accumulation 4

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ \ newkey),\ \neg Know(Y,\ \ newkey)}}{\boldsymbol{P \mid\sim_{\alpha\ o\ Generate(X,\ newkey)}\ \neg Q}}$$

This means that, if X alters the cryptographic keys k or $<Spb(X), Spv(X)>$ or $<Kpb(X), Kpv(X)>$, and does not let Y know the *newkey* that is created by X, then we can say non-monotonically that the conclusion Q which has been proved to be true is not correct any more after the action α *o* *Generate(X, newkey)*, which consists of *Generate*() and *Send*(), and so forth.

In circumstance (2), corresponding rules are listed below.

(R-II-1) Accumulation 5

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ m),\ Know(Y, <m',\ \ m>),\ Equal(m,\ \ m')}}{\boldsymbol{P \vdash_{\alpha\ o\ Generate(X,\ m)}\ Q}}$$

That is, if X creates a new message m and let Y know m and m' since the principals have no memory to m', then, if the contents of m remain the same as the message m' that was formerly produced, the conclusion Q still stays true after the action α *o* $Generate(X, m)$. Thus, message m represents the combination of $m_1, m_2, \ldots,$ *and* m_n.

(R-II-2) Accumulation 6

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ m),\ \neg Know(Y, <m',\ \ m>)}}{\boldsymbol{P \mid\sim_{\alpha\ o\ Generate\ (X,\ m)}\ \neg Q}}$$

That is, if X creates a new message m, but does not let Y know what has been changed about m', then we can say non-monotonically that the conclusion Q does not remain true after the action α *o* $Generate(X, m)$.

For circumstance (3), several rules are described as follows.

(R-III-1) Accumulation 7

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X, <newkey,\ m>),\ Know(Y,\ \ <newkey,\ m,\ m'>),\ Equal(m,\ \ m')}}{\boldsymbol{P \vdash_{\alpha\ o\ Generate(X,\ <newkey,\ m>)}\ Q}}$$

This means that, if principal X alters the cryptographic keys, k or $<Spb(X)$, $Spv(X)>$ or $<Kpb(X)$, $Kpv(X)>$, and the message, and lets Y know the *newkey*, m' and m, then, if the contents of message m produced by X correspondingly remains the same as the message m' which was created originally, then the conclusion Q still stays true after the action α *o* $Generate(X, <newkey, m>)$.

(R-III-2) Accumulation 8

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ <newkey,\ m>),\ Know(Y, <newkey,\ m,\ m'>),\ \neg Equal(m,\ m')}}{\boldsymbol{P \mid\sim_{\alpha\ o\ Generate(X,\ <newkey,\ m>)} \neg Q}}$$

This means that, during the execution of the protocol, if X alters the cryptographic keys, k or $<Spb(X)$, $Spv(X)>$ or $<Kpb(X)$, $Kpv(X)>$, and the message, and lets Y know the *newkey*, m' and m, then, if the contents of message m that is produced currently by principal X do not keep the same as message m', then we can say non-monotonically that the conclusion Q does not keep true after action α *o* $Generate(X, <newkey, m>)$.

(R-III-3) Accumulation 9

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ <newkey,\ m>),\ \neg Know(Y,\ newkey)}}{\boldsymbol{P \mid\sim_{\alpha\ o\ Generate\ (X,\ <newkey,\ m>)} \neg Q}}$$

This means that, during the execution of the protocol, if X alters the cryptographic keys, k or $<Spb(X)$, $Spv(X)>$ or $<Kpb(X)$, $Kpv(X)>$, and the message, and does not let Y know the *newkey*, then we can say non-monotonically that the conclusion Q does not stay true after the action α *o* $Generate(X,<newkey, m>)$, which consists of $Generate()$ and $Send()$, and so forth.

(R-III-4) Accumulation 10

$$\frac{\boldsymbol{P \vdash_{\alpha} Q,\ Generate(X,\ <newkey,\ m>),\ \neg Know(Y,\ <m',\ m>)}}{\boldsymbol{P \mid\sim_{\alpha\ o\ Generate\ (X,\ <newkey,\ m>)} \neg Q}}$$

This means that, during the execution of the protocol, if X alters the cryptographic keys, k or $<Spb(X)$, $Spv(X)>$ or $<Kpb(X)$, $Kpv(X)>$, and the message, and does not let Y know what has been changed about the message m', then we can say non-monotonically that the conclusion Q does not stay true after the action α *o* $Generate(X, <newkey, m>)$, which consists of $Generate()$ and $Send()$, and so forth.

(R-III-5) Accumulation 11

$$\frac{\boldsymbol{P \vdash_{\alpha} Q},\ \boldsymbol{Generate(X,\ \ newkey)},\ \boldsymbol{\neg Know(Y,\ \ newkey)}}{\boldsymbol{P \mid\sim_{\alpha\, o\, Generate(X,\ newkey)} \neg Q}}$$

This means that, during the execution of the protocol, if X alters the cryptographic keys, k or $<Spb(X),\ Spv(X)>$ or $<Kpb(X),\ Kpv(X)>$, and does not let Y know the *newkey*, then we can say non-monotonically that the conclusion Q does not reamin true after the action $\alpha\ o\ Generate(X,\ newkey)$.

There is still one more possible situations in circumstance (3). This has been introduced at the end of R-I-1. In this instance, R-I-1 can be used to describe it.

For circumstance (4) two rules are generated accordingly:

(R-IV-1) Accumulation 12

$$\frac{\boldsymbol{P \vdash_{\alpha} Q},\ \boldsymbol{Generate(X,\ m)},\ \boldsymbol{Equal(m,\ \ m')}}{\boldsymbol{P \vdash_{\alpha\, o\, Generate(X,\ m)}\ Q}}$$

This means that, during the execution of the protocol, if X alters message m, but the contents of message m that is produced at the same time by principal X remain the same as message m', then we can say that the conclusion Q stays true after action $\alpha\ o\ Generate(X,\ m)$, since the principals have memory to message m'.

(R-IV-2) Accumulation 13

$$\frac{\boldsymbol{P \vdash_{\alpha} Q},\ \boldsymbol{Generate(X,\ m)},\ \boldsymbol{\neg Equal(m,\ \ m')}}{\boldsymbol{P \mid\sim_{\alpha\, o\, Generate(X,\ m)} \neg Q}}$$

This means that, during the execution of the protocol, if X alters message m, but the contents of message m that is produced at the same time by principal X does not keep the same as message m', then we can say non-monotonically that the conclusion Q does not stay true after action $\alpha\ o\ Generate(X,\ m)$ since the principals have memory to message m'.

The accumulation rules synthetically described above take into account the possibility that might happen with regard to practical e-commerce. Meanwhile, it is also necessary to point out some of the disadvantages of certain notations, such as the expression of cryptographic keys. It would appear that these notations might never be altered after designating the principal X. In fact, they may be altered constantly, along with the change of time, so if we want the definition of these rules to become more accurate, the current notation system needs to be improved further. In this chapter, it assumes that all the notations keep the same content and meaning as the original. Exceptions to this will be discussed in future work.

3.5.3 Inference Format

This inference format stems from the accumulation property described above where ε, α_1, α_2, ..., α_n and f_{ij} express a sequence of actions and a set of formulae, respectively.

$$\begin{array}{ll} \varepsilon \text{ (empty sequence of action)} & f_{01},\, f_{02} \ldots f_{0m_0} \\ \alpha_1 & f_{11},\, f_{12} \ldots f_{1m_1} \\ \alpha_2 & f_{21},\, f_{22} \ldots f_{2m_2} \\ & \vdots \\ \alpha_n & f_{n1},\, f_{n2} \ldots f_{nm_n} \end{array}$$

The above expressions can be compressed by using an assertion to describe the procedure of deduction. The union of formulae is derived from the definition of the assertion. For instance, $\{f_{01},\, f_{02} \ldots f_{0m_0}\}$ is a formula, so each f_{0i} $(0 \leq \mathrm{i} \leq m_0)$ is a subset of this formula. Furthermore, the synthesis of assertion is, in fact, derived from inference rules R-3 and R-4. This format can be described by another expression as shown below:

$$\begin{array}{c} \{f_{01},\, f_{02} \ldots f_{0m_0}\} \vdash \alpha_1 \quad \{f_{11},\, f_{12} \ldots\ldots f_{1m_1}\} \\ \{f_{01},\, f_{02} \ldots f_{0m_0}\} \vdash \alpha_1 \cdot \alpha_2\ \{f_{21},\, f_{22} \ldots\ldots f_{2m_2}\} \\ \vdots \\ \{f_{01},\, f_{02} \ldots f_{0m_0}\} \vdash \alpha_1 \cdot \alpha_2 \cdot \ldots\ldots \cdot \alpha_n\ \{f_{n1},\, f_{n2} \ldots\ldots f_{nm_n}\} \end{array}$$

This inference format is based on the accumulation and dynamic properties of security protocols. It is well known that a message can be modified or intercepted during transmission, so an eavesdropper can impersonate the sender or receiver and continue the transaction. In this chapter, it assumes that these problems can be detected and are protected by existing technologies such as [40, 70, 92]. Therefore, we do not need to be too concerned about the network communication, eavesdropping, hashing and encryption algorithms. Thus, this chapter focuses on how the ENDL may be used for the verification of secure transaction protocols, rather than how to construct a secure protocol.

3.5.4 Verification Instances of Security Protocols in ENDL

To evaluate the framework ENDL, three instances are used to demonstrate the use of the ENDL when verifying a security protocol.

Example 3.1. **Distribution of Public keys in Needham and Schroeder's protocol.**

A timestamp can be added to Needham and Schroeder's protocols for public key systems. This was proposed by Denning [40]. In this instance, it shows that our logic is also able to detect a known flaw in the protocol.

The protocol opens with A consulting the authentication server AS in the clear to find B's public key. The exchange can be described in a series of actions.

$$A \rightarrow AS\text{: } \alpha_1 = Send(A, AS, <A, B>) \tag{1.1}$$
$$AS \rightarrow A\text{: } \alpha_2 = Send(AS, A, S(<B, Spb(B)>, Spv(AS))) \tag{1.2}$$
$$A \rightarrow B\text{: } \alpha_3 = Send(A, B, E(<ID_A, A>, Spb(B))) \tag{1.3}$$
$$B \rightarrow AS\text{: } \alpha_4 = Send(B, AS, <B, A>) \tag{1.4}$$
$$AS \rightarrow B\text{: } \alpha_5 = Send(AS, B, S(<Spb(A), A>, Spv(AS))) \tag{1.5}$$
$$B \rightarrow A\text{: } \alpha_6 = Send(B, A, E(<ID_A, ID_B>, Spb(A))) \tag{1.6}$$
$$A \rightarrow B\text{: } \alpha_7 = Send(A, B, E(ID_B, Spb(B))) \tag{1.7}$$

where $Spv(AS)$ denotes AS's private signature key and $Spb(B)$ is B's public signature key. A is presumed to know AS's public signature key (derived from axiom 2-2) since AS is an authority.

We start the verification from the goal $Auth(A, AS, Spb(B))$ and let CK=<B, $Spb(B)$>. Formula P denotes the set of premise; α is the combination of a series of action; and formula Q denotes the object we want to verify.

$$P = \{Know(A, Spb(AS))\},$$
$$\alpha = \{Generate(A, <A, B>)\ o\ \alpha_1\ o\ Generate(AS, S(CK, Spv(AS)))\ o\ \alpha_2\ o\ Generate(A, E(<ID_A, A>, Spb(B)))\ o\ \alpha_3\}$$
$$Q = \{Auth(A, AS, Spb(B))\}.$$

Proof:

(1)	$Know(A, Spb(AS))$	[2-2]
(2)	$Generate(A, <A, B>)$	[action]
(3)	$Know(A, <A, B>)$	(2)[R-2]
(4)	$Send(A, AS, <A, B>)$	(3)[1.1]
(5)	$Know(AS, <A, B>)$	(4)[R-1]
(6)	$Know(AS, B)$	(5)[6-1]
(7)	$Generate\ (AS, S(CK, Spv(AS)))$	(6)[2-4][action]
(8)	$Know(AS, S(CK, Spv(AS)))$	(7)[R-2]
(9)	$Send(AS, A, S(CK, Spv(AS)))$	(8)[1.2]
(10)	$Know(A, S(<B, Spb(B)>, Spv(AS))$	(9)[R-1]
(11)	$\rightarrow Auth(A, AS, Spb(B))$	(1)(3)(10)[5-3][R-5]

The final conclusion is that we cannot reach the goal $Auth(A, AS, Spb(B))$ since the existing conditions do not satisfy the requirement of axiom (5-3). According to the non-monotonic rule (R-5), we may conclude non-monotonically

that A does not believe the validity of $Spb(B)$ encrypted by AS. The reason this flaw works is because the intruder was able to intercept the message from (4) and (9) and replay the encrypted component from message (10). The weakness that allowed the attack lies in the fact that this protocol did not add a timestamp to step (9). This would have made the recipient suspect that the message was a replay [40].

Due to the non-monotonic property, the authentication here contains only eleven steps, thereby ENDL is more efficient than other logics.

Example 3.2. **Case 1 of Cardholder Registration in SET protocol.**

From SET protocol, a simple process of registration is intercepted. It starts when the cardholder C receives an initial response sent by CA. There are two principals in this instance: cardholder C and certificate authority CA. CA, in fact, consists of several levels of certificate authority, as described in Figure 3.2. In this instance, some of the new notations will be used to describe this particular verification.

$$CA \rightarrow C\text{: } \alpha_1 = Send(CA,\ C,\ <CertS(CA),\ CertK(CA),\ InitRes,\ S(H(InitRes),\ Spv(CA))>) \tag{2.1}$$

$$C \rightarrow CA\text{: } \alpha_2 = Send(C,\ CA,\ <e(RegFormReq,\ k),\ E(<PAN,\ k>,\ Kpb(CA))>) \tag{2.2}$$

CA generates response $InitRes$ and digitally signs it. It then sends the response along with the CA certificate to C. After verifying the CA certificate, C generates registration form $RegFormReq$ and encrypts the message with a randomly generated symmetric key k. This key, along with the C's primary account number (PAN), is then encrypted with $Kpb(CA)$. Only the CA, C, and the Issuer know PAN, which is effectively obfuscated by using a blinding technique [19]. Then C transmits these messages to CA; CA decrypts key k and the cardholder's PAN with $Kpv(CA)$. It then decrypts the $RegFormReq$, using k. Eventually, CA determines the appropriate registration form, and digitally signs it by using $Spv(CA)$. It then sends them to C.

The verification starts with the goal $Auth(C,\ CA,\ <Spv(CA),\ Spb(CA)>)$ and $Auth(C,\ CA,\ <Kpv(CA),\ Kpb(CA)>)$. The definitions of P, α and Q have the same meaning as above.

$$\begin{aligned}
&P = \{Know(C,\ Spb(CA))\},\\
&\alpha = Generate(C,\ InitRes)\ \mathrm{o}\\
&\quad Generate(CA,\ <CertS(CA),\ CertK(CA),\ InitRes,\\
&\quad S(H(InitRes),\ Spv(CA))>)\ \mathrm{o}\ \alpha_1\ \mathrm{o}\ Verify(C,\ CertS(CA),\ CA)\ \mathrm{o}\\
&\quad Verify(C,\ CertK(CA),\ CA)\ \mathrm{o}\ Generate(C,\ RegFormReq)\ \mathrm{o}\\
&\quad Generate(C,\ PAN)\ \mathrm{o}\ Generate(C,\ k)\ \mathrm{o}\ \alpha_2\\
&Q = \{Auth(C,\ CA,\ <Spb(CA),\ Spv(CA)>),\ Auth(C,\ CA,\ <Kpb(CA),\\
&\quad Kpv(CA)>),\ Auth(C,\ CA,\ InitRes)\}
\end{aligned}$$

Proof:

(1) $Know(C, Spb(CA))$ [2-2]
(2) $Generate(CA, InitRes)$ [action]
(3) $Know(CA, InitRes)$ (2)[R-2]
(4) $Generate(CA, <CertS(CA), CertK(CA), InitRes, S(H(InitRes), Spv(CA))>)$ (3)[2-4][action]
(5) $Know(CA, <CertS(CA), CertK(CA), InitRes, S(H(InitRes), Spv(CA))>)$ (4)[R-2]
(6) $Send(CA, C, <CertS(CA), CertK(CA), InitRes, S(H(InitRes), Spv(CA))>)$ (5)[2.1]
(7) $Know(C, <CertS(CA), CertK(CA), InitRes, S(H(InitRes), Spv(CA))>)$ (6)[R-1]
(8) $Know(C, CertS(CA))$ (7)[6-1]
(9) $Know(C, CertK(CA))$ (7)[6-1]
(10) $Know(C, S(H(InitRes), Spv(CA)))$ (7)[6-1]
(11) $Verify(C, CertS(CA), CA)$ [abbreviation]
(12) $IsVerified(C, CertS(CA))$ [discriminant]
(13) $Auth(C, CA, <Spb(CA), Spv(CA)>)$ (8)(11)(12)[7-1]
(14) $Verify(C, CertK(CA), CA)$ [abbreviation]
(15) $IsVerified(C, CertK(CA))$ [discriminant]
(16) $Auth(C, CA, <Kpb(CA), Kpv(CA)>)$ (9)(14)(15)[7-2]
(17) $\rightarrow Auth(C, CA, InitRes)$ (1)(7)(10)[6-1][Theorem 3.1][R-5]

From steps (11) to (16), if the cardholder fails to validate the *CA*'s signature certificate or key-exchange certificate, the remaining verification will be stopped automatically. The certificate chain is not applied in NDL, but it is a very important component in Cardholder Registration. For the verification of security protocols, the certificate chain is too complicated to describe here. Meanwhile, if without passing the validation of certificate, we cannot ensure that the private key used to sign a process is associated with the right certificate authority, then we add this process to strengthen verification and make it more compendious and reliable. During this verification, if *C* fails to authenticate the validity of *InitRes* sent by *CA*, then the authentication should be halted immediately owing to the non-monotonic rule (R-5), namely *fail-negate*. Certainly, *C* will not send the registration form request to *CA*. The flaw detected for *InitRes* encrypted by $Spv(CA)$ does not include the timestamp and identifier, so the intruder can replay this message in a later transaction. This problem is common in the literature (For example in Syverson, 1994 [148]). The identifier is a random number and is used only once, but the certificate is a long-term word. Actually, certificate cannot be relied upon to be absolutely secure so the certificate is not a substitute for the identifier.

(1) $C \rightarrow Z(CA)$: *InitReq*
(2) $Z(C) \rightarrow$CA: *InitReq*′

(3) $CA \rightarrow Z(C)$: $<CertS(CA),\ CertK(CA),\ InitRes',\ S(H(InitRes'),\ Spv(CA))>$
(2′) $Z(C) \rightarrow CA$: $InitReq''$
(3′) $CA \rightarrow Z(C)$: $<CertS(CA),\ CertK(CA),\ InitRes'',\ S(H(InitRes''),\ Spv(CA))>$
(4) $Z(CA) \rightarrow C$: $<CertS(CA),\ CertK(CA),\ InitRes',\ S(H(InitRes'),\ Spv(CA))>$

Here, *InitReq* denotes the initial request sent by C to CA. The intruder Z intercepts the initial request from C to CA and replaces it with a new initial request $initReq'$. It then sends the result to CA as message (2). CA replies with message (3). Z impersonates C to produce a new message (2′) and sends it to CA. CA answers C with a corresponding message (3′), and then Z intercepts it. At last, Z impersonates CA to send an outdated message (4) that is intercepted by Z from message (3). C cannot authenticate the message since it does not include a timestamp and identifier. This attack is used continuously until C wants to bring about authentication between C and CA again. An optional solution is to include timestamp and identifier in the message sent by CA, even though you think certificate is secure. In fact, the protocol designer must be very careful to detect every possible subtle drawback when the protocol is in the design stage. Otherwise, if a protocol with flaws is placed in practical environment, such as instant stock-trading. It can cause a big economic loss if an intruder detects the flaws.

Example 3.3. **Case 2 of Cardholder Registration.**

$$C \rightarrow CA: \alpha_1 = Send(C, CA, e(<m, S(H(m), Spv(C))>, k_3)) \quad (3.1)$$
$$C \rightarrow CA: \alpha_2 = Send(C, CA, E(<k_3, AcctInf>, Kpb(CA))) \quad (3.2)$$

After receiving the message sent by cardholder C, CA decrypts k_3 and account information *AcctInf* using *Kpv(CA)*. Then it decrypts *RegFormReg*, using k_3; and CA determines the appropriate registration form *RegForm*, and digitally signs it with *Spv(CA)*. It then sends the registration form and $CertS(CA)$ to cardholder C. Let $m = <CertReq,\ k_2\ ,\ Spb(C)>$.

There are two principals, C and CA, in this transaction. After validating the registration form sent by CA, cardholder C creates one pair of signature keys, $Spv(C)$ and $Spb(C)$. Meanwhile, C also creates two new symmetric key, k_2 and k_3, and the certificate request *CertReq*, including the information entered the registration form. Generally, it assumes that they do not alter the keys and have memory to the message. However, we cannot exclude the possibility that someone might modify the keys if: (1) the principal doubts that the key may have been used to be replays by an eavesdropper, (2) the transaction fails, and (3) the key is too old to use. On the other hand, the principal may

also have no memory to the message it sent or received previously. Thus, the former accumulation rule must be classified to verify possible security flaws. The related keys and the message generated in this process are listed below:

Cryptographic keys: $Spv(C)$, $Spb(C)$, k_2, k_3
Message: *CertReq*

Based on the above keys and message, the following modifications are perhaps conducted:

- C alters cryptographic keys, and notifies CA promptly before applying the new keys.
- C alters them, but does not notify CA before applying the new keys.
- C generates *CertReq* again, and it stays the same as the old one.
- C generates *CertReq* again, but it does not keep the same as the old one.
- C alters cryptographic keys, and notifies CA before applying the new keys; C generates *CertReq* again, and it keeps the same as the old one.
- C alters cryptographic keys, and notifies CA before applying the new keys; C generates *CertReq* again, but it does not keep the same as the old one.
- C alters cryptographic keys, but does not notify CA before applying the new keys; C generates *CertReq* again, and it keeps the same as the old one.
- C alters cryptographic keys, but does not notify CA before applying the new keys; C generates *CertReq* again, but it does not stay the same as the old one.

The first and second cases assume the principal has memory to message. In contrast, we assume that the principal has no memory to message in the remaining items. In the first instance, if C alters the cryptographic key, $Spv(C)$, $Spb(C)$, k_2, k_3, and notifies CA before applying the new key, CA can make the older key invalid and stop using them. Meanwhile, because the principal has memory to message, we still believe that the result remains true after a series of new actions using the (R-I-1) to verify the process. However, if C does not allow CA to know the new key, the former result cannot be authenticated, even though the principal has memory to message. This may damage the security of transactions. For the other items, verification can be developed according to the corresponding accumulation rules. The procedure for verification should be the same as in the first instance. Detailed verification will be showed in our future work. Therefore, only the second instance is used to illustrate the usefulness of classified accumulation rules.

$$P = \{Know(C, Kpb(CA)), Know(C, AcctInf)\},$$
$$\alpha = Generate(C, <Spv(C), Spb(C)>) \circ Generate(C, CertReq) \circ Generate(C, k_2) \circ Generate(C, k_3) \circ \alpha_1 \circ \alpha_2$$
$$Q = \{Auth(CA, C, m)\}$$

Proof:

(1)	$Know(C,\ Spb(CA))$	[2-2]
(2)	$Generate(C,\ <Spv(C),\ Spb(C)>)$	[action]
(3)	$Know(C,\ <Spv(C),\ Spb(C)>)$	(2)[R-2]
(4)	$Generate(C,\ CertReq)$	(3)[2-4][action]
(5)	$Know(C,\ CertReq)$	(4)[R-2]
(6)	$Generate(C,\ <k_2,\ k_3>)$	[action]
(7)	$Know(C,\ <k_2,\ k_3>)$	(6)[R-2]
(8)	$Know(C,\ <CertReq,\ k_2,\ Spb(C)>)$	(3)(5)(7)[6-1]
(9)	$Know(C,\ H(m))$	(8)[4-1]
(10)	$Know(C,\ S(H(m),\ Spv(C)))$	(3)(9)[6-1]
(11)	$Know(C,\ e(<m,\ S(H(m),\ Spv(C))>,\ k_3))$	(7)(8)(10)[6-1][1-1]
(12)	$Know(C,\ AcctInf)$	[premise]
(13)	$Know(C,\ E(<k_3,\ AcctInf>,\ Kpb(CA)))$	(7)(12)[6-1][2-1]
(14)	$Send(C,\ CA,\ e(<m,\ S(H(m),\ Spv(C))>,\ k_3))$	[action]
(15)	$Send(C,\ CA,\ E(<k_3,\ AcctInf>,\ Kpb(CA)))$	[action]
(16)	$Know(CA,\ e(<m,\ S(H(m),\ Spv(C))>,\ k_3))$	(14)[R-1]
(17)	$Know(CA,\ E(<k_3,\ AcctInf>,\ Kpb(CA)))$	(15)[R-1]
(18)	$Know(CA,\ m)$	(30)(17)[2-3][3-2][3-1]
(19)	$Know(CA,\ S(H(m),\ Spv(C)))$	(16)(17)[2-3][3-2][3-1]
(20)	$\rightarrow Auth(CA,\ C,\ m)$	

The step (20) implies two possibilities: one is that the message m does not include a timestamp and identifier, thus CA cannot authenticate C on message m due to Theorem 3.1 and R-5; the other is that cardholder C generates new cryptographic keys and does not notify CA, thus we can conclude CA cannot authenticate C on message m for (R-I-4), even though CA weakly authenticates C on message m in the first case. The second case cannot be carried out by the NDL. Thereby, the ENDL greatly strengthens the verification and should be a beneficial supplement to the NDL.

Comparing with other logics, ENDL is more compatible with existing security protocols due to the dynamic and non-monotonic properties. Also, the strict authentication framework of ENDL helps us detect some subtle flaws that might be easily ignored in other logics. With these three instances, we can demonstrate that the ENDL is effective for verifying security protocols. Example 3.1, derived from Needham and Schroeder's protocols for public key systems, is regarded as secure. Although author uses double handshake to assure the information remains secret during the communication, we have proved it has a flaw. Example 3.2 is drawn from SET protocol that is commonly thought to be a standard for future e-commerce. In our method, we detect a subtle flaw there. The problems are not serious since it is not easy for an intruder to turn the content of the certificate. However, this certainly represents an attack since it leads to an intruder holding an incorrect belief:

CA believes that it is C who thought C was talking to CA. Example 3.3 also comes from the Cardholder Registration of the SET protocol. By validating the certificate request of C in the new accumulation rules, a new subtle flaw has been detected. This may be neglected in the NDL, although it is a potential threat to the security of SET. To our knowledge, it is the first time to uncover these flaws in cardholder registration.

3.6 Summary

E-commerce has played a very important role in global economic growth today. At the same time, however, its evolution has resulted in the increase of both the vulnerability of e-commerce systems to security violations and the damage that such violations may cause.

A number of secure transaction protocols have been developed to ensure secure transactions. However, these secure protocols may be subject to diverse malicious attacks. Regardless of much efforts on improving the protocols, there are still subtle flaws found from insecure protocols. A variety of formal methods have been developed to analyse secure protocols in e-commerce systems. However, their abilities are still far from what we expected.

Traditional formal methods, such as BAN logic [22] and GNY logic [62], have showed their limitations in analysing secure transaction protocols. Although a number of formal approaches have been developed for formal analysis of e-commerce protocols, they are either complicated to use or incapable of being extended to other protocols.

In this chapter, ENDL is proposed for facilitating the verification of secure transaction protocols. It is combined with dynamic and non-monotonic properties, and presents more challenges than existing logics used in the analysis of secure transaction protocols.

In particular, the verification automatically halts in answer to any unsuccessful authentication, and therefore we can detect the defects of security protocols in advance. In addition, by using the notation of identifier and timestamp that have proved to be efficient in helping the designer of protocol to detect attacks, we can protect information against replays. The proposed methodology satisfies the requirements of accumulation rules. To make it more reasonable and applicable, the fundamental accumulation rule is classified further to generate thirteen new rules in terms of corresponding circumstances.

The examples in Section 3.5.4 have shown that the ENDL is useful for finding some well-known flaws in security protocols. Meanwhile, it also detects some subtle flaws in the SET protocol, which was previously believed to be secure [7]. Therefore, the framework ENDL can be a useful complement to existing formal analysis of electronic transaction protocols.

4

Model Checking in Security Protocol Analysis

Theorem proving and model checking are two main approaches used for the formal analysis of security protocols. As described in Chapter 1, theorem proving focused on the verification of authentication protocols and cryptography protocols. Although Heintze [68] firstly used model checking to analyse electronic transaction protocols, the efforts used for model checking of electronic transaction protocols are underdeveloped due to increasing complexity and varied types of application of the protocols.

Model checking is a technique for verifying finite state concurrent systems such as hardware design and communication protocols. Specifications of the systems are represented as temporal logical formulae, and efficient symbolic algorithms are applied to convert the model defined by the systems and check if the specification holds or not. In contrast to traditional approaches that are based on simulation, testing and deductive reasoning, model checking is automatic and usually fast. On the other hand, if an error is found, model checking is able to produce a counterexample that shows the source of the error.

There have been a number of model checking methods developed for the analysis of e-commerce protocols. Ray [75] shows how model checking can be used to obtain an assurance about the existence of the properties in an e-commerce protocol such as money atomicity and validated receipt. The model checker can be used to evaluate what failures cause the violation of one or more of the properties. In [157], algorithms and rules are developed to translate visually modelled e-commerce protocols into formal models that are then verified using an extended UML formalism. This approach is applied to the design of an e-commerce protocol NetBill. An extended fair-exchange standard is described in [12], which includes atomicity assurance and uses model checking to verify the correctness of the implementation of e-commerce protocols. A main challenge in model checking is dealing with the state exploration problem. To alleviate this problem, a parallel model checker [78] is proposed to analyse a security protocol that was developed to facilitate secure and fair exchange. In particular, the model checking based on FDR is used to analyse

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 107–129, 2008.

the SET protocol and check whether five essential correctness properties are satisfied [100].

In contrast to security protocols that involve secrecy and authentication, the correctness conditions for electronic transaction protocols contain more components. These conditions present interesting challenges for the traditional theorem proving. Furthermore, the number of principals and data are unforseen. To address these problems, a verification model based on ENDL is recently developed by us [28].

It is usually faster than theorem proving, and extensible because the fundamental security mechanisms of different security protocols remain unchanged. For brevity, some abstractions are employed, such as the low-level details of the underlying cryptographic mechanisms. Thus, we could turn our sight on the verification of security properties we expected to hold. Several examples are validated by using this model. From the observation, the verification model is a useful complement to the traditional theorem proving in verifying security protocols.

In Section 4.1, it outlines the current model checking approaches. Section 4.2 describes the components and design of the verification model. Several instances are then validated by using this model. In Section 4.3, it compares the verification model to theorem proving. We discuss the other model checkers that can be used to analyse security protocols in Section 4.4. Section 4.5 gives a summary to this chapter.

4.1 An Overview of Model Checking in Analysing E-Commerce Protocols

Model checking is techniques that formally specify the system as logical formulae, and efficient symbolic algorithms are used to traverse the model defined by the system and check whether the specification holds or not. Extremely large state-spaces can often be traversed in a short time. In contrast to the traditional theorem proving, model checking is automatic and usually fast.

Usually, the users want to ask the following questions when checking their design requirements:

- Do they match the user's needs or reflect the user's requirements?
- Are the requirements clear and readable?
- Are the requirements flexible or realizable for design and development?
- Are the requirements written in an abstract way, so as to leave sufficient freedom to the designers and developers to implement them?

Regardless of some helps from modelling tools such as UML, the main process has to be completed manually. To improve the quality of the requirements, it is critical and fundamental to use an unambiguous and rigorous formal

language for describing the requirements. A model checking tool accepts system requirements (called *models*) and a system property (called *specification*) that the system is excepted to satisfy. It answers *yes* if the model satisfies given specifications and generates a counterexample otherwise. By studying the counterexample, we can identify the error in the model and correct the model. The sufficiently satisfied system properties can increase the confidence in the correctness of the model.

Model checking has been successfully used in hardware validation [69, 108]. Also, model checking has recently attracted the attention of the software designer [152, 158]. In security domain, a number of works have been done. Lowe [98] used FDR to debug and validate the correctness of Needham-Schroeder protocol and Heintze [68] used FDR to verify the electronic transaction protocols, including NetBill and a simplified digital cash protocol. Protocol verification aims at proving that protocols meet their specifications, i.e., that the actual protocol behaviour coincides with the desired one. On the other hand, if the logic formulae cannot be proved within the finite state, some feasible suggestions will be generated and sent to the user. However, as to our knowledge, not much work has been found to analyse electronic transaction protocols using model checking due to their complexity. They may use different formal languages to model various protocols. Thus, it is usually infeasible to transfer an approaches to analyse a variety of protocols.

4.1.1 Model Checking for Failure Analysis of Protocols

The wide application of the world wide web has resulted in the popularity of e-commerce. Many e-commerce protocols have been proposed to ensure secure transactions without unauthorized disclosure and modification. A few desirable properties of protocols, such as money atomicity and goods atomicity, and validated receipt have been identified by researchers. However, it is a key issue to verify whether a given protocol satisfies these properties in the presentence of site and/or communication failures.

An e-commerce protocol proposed by Ray [76] is designed for electronic transactions including digital products. Let C, M and TP be a customer, a merchant and a trusted third party. The protocol can be formally described below in an abstract way.

1. $TP \rightarrow C$: download of product encrypted with key K_1
2. $C \rightarrow M$: purchase order
3. $M \rightarrow C$: product encrypted with a second key K_2
4. $M \rightarrow TP$: the decrypting key K for the product and the approved purchase order
5. $C \rightarrow TP$: the payment token and a copy of the purchase order
6. $TP \rightarrow C$: the decrypting key
7. $TP \rightarrow M$: payment token

Actually, the messages exchanged are signed, encrypted and sent along with a cryptographic checksum, whereas the cryptographic aspects of the protocols are abstracted away when modelling the protocol.

In this approach, Failure Divergence Refinement (FDR) model checker is applied. The analysed protocol is represented as a communicating sequential process (CSP), called *system*. The property that we want to check is expressed as another CSP process, called *SPEC*. If the set of behaviours generated by *SYSTEM* is a subset of those generated by *SPEC*, we can say that the protocol satisfies the property.

Modelling the communication between the processes. A sender sends messages over a unique channel (the sender's *out* channel for a particular receiver) while the receiver receives the message over another channel (the receiver's *in* channel). For each pair of channel, a process is needed to read data from the *out* channel and write the data into the *in* channel. For example, four channels including *minc*, *coutm*, *cinm* and *moutc*, and two processes including COMMmc and COMMcm are involved to model the communication between the merchant and the customer.

Modelling the customer, merchant and the trusted third party processes. Initially, the customer waits for an encrypted product from the trusted third party.

$$\text{CUSTOMER} = \text{cint } ?x \rightarrow \text{DOWNLOADED\_EGOODS}(x)$$

where *cint* represents a communication channel between customer and the third party.

A purchase order is sent to M once C has downloaded the product, which is modelled as:

$$\text{DOWNLOADED\_EGOODS}(x) = \text{coutm !po} \rightarrow \text{PO\_SENT}(x)$$

where *po* represents the purchase order, which is the set of all data communicated from the customer to the merchant directly .

The next step involves checking the encrypted product sent by the merchant and comparing the encrypted product from the merchant and those from the third party. A payment token will be sent to the third party when the customer is satisfied with the encrypted product. After sending the payment, the third party sends the customer the key or an abort message.

Modelling the merchant process. On the merchant side, the merchant waits to receive a purchase order from the customer.

$$\text{MERCHANT} = \text{minc } ?x \rightarrow \text{if } (x \text{ PO\_SENT}(x))$$

In response to the purchase order, the merchant sends an encrypted product to the customer and the decryption key to the trusted third party. The merchant then waits to receive the payment token from *TP*.

Modelling the trusted third party. The trusted third party sends the encrypted product to the customer and waits to receive the payment token from the customer and the key from the merchant. This is modelled as:

$$\text{TP} = \text{toutc !encryptedGoods1} \rightarrow \text{WAIT\_TOKEN\_KEY}$$

After receiving both the key and payment token and validating their correctness, *TP* proceeds the sending out the key to the customer and the token to the merchant.

In addition to the modelling of participants, some desirable properties including money atomicity, goods atomicity and validated receipt are modelled. For example, the money atomicity is satisfied when (i) the customer sends the token and the merchant receives it or (ii) the customer sends the token and receives a transaction abort message. This is modelled as:

$$\text{SPEC1} = \text{STOP} \mid \sim \mid ((\text{coutt.paymentToken} \rightarrow \text{mint.paymentToken} \rightarrow \text{STOP}) \; [] \; (\text{coutt.paymentToken} \rightarrow \text{cint.transAborted} \rightarrow \text{STOP}))$$

In the similar way, the good atomicity and the validated receipt property can be modelled.

The model checker aims to detect the failures that do not preserve the desirable properties of money atomicity, goods atomicity, and validated receipt. The data may get lost in an unreliable communication channel. The customer, merchant and third trusted party should be able to abort at certain points in the protocol without violating the properties. A protocol failure analysis is conducted to see why the protocol is failure resilient. According to the analysis, the authors proposed a mechanism that preserves the properties even in the event of sites or communication failures.

Another similar work can be seen in [12]. It delineates an extended fair-exchange standard and represents how a model checker using FDR can facilitate verification of a standard's atomicity requirement. In particular, the verification prevents the failure of protocols when unforeseen circumstances occur.

4.1.2 Automatic Analysis of E-commerce Protocols Using UML

The Unified Modeling Language (UML) has been widely applied in supporting object-oriented software development. In recent years, many researchers have extended UML for design and development of e-commerce protocols. However, the existing formalisms are insufficient to satisfy the requirement of developing e-commerce protocols. A novel model checking technique that integrates the refined AUML [157] protocol diagrams is introduced to exploit the automated analysis of e-commerce protocols.

AUML is a subset of an agent-based extension to the standard UML for the specification of *agent interaction protocols* (AIP). The particular extension includes lifeline and messaging. Furthermore, a process for e-commerce protocols is developed. The modelling of e-commerce protocols using refined protocol diagrams consist of the following steps:

Refinement of AUML protocol diagrams. The absence of definite semantics of UML makes it ambiguous and therefore difficult to be interpreted mechanically. The *Refined Protocol Diagram* (*RPD*), is going to meet the requirements for e-commerce protocols design and also satisfy our tastes for visual modelling. The refinements about messages passing are listed below:

- *Message broadcast* corresponds to the message passing with constrain $\{1 \cdots n\}$, which represents that the message is sent n times simultaneously.
- *Message triggering* implies that some internal events trigger the message sending.
- *Message synchronization* has the similar blocked semantics with single message receiving. The difference is merely that some messages arrival has to wait for.
- *Constraint* "*silent*" represents that message sending or receiving have no effect on the current state of role.
- *Message causality* is introduced to causally relate two messages on the same lifeline.

Textual notations for the refined AUML protocol diagram. Formal analysis of protocols always require a form of description that is tractable by algorithms. A textual version of the refined protocol diagrams is defined. In refined protocol diagrams, each role is mapped to a pattern of interaction behaviour, which includes interaction elements and operators. The syntax of interaction behaviour of one role is defined as:

$$\text{Beh} ::= \text{Ele} \mid \text{Beh}_i \rightarrow \text{Beh}_j \mid \text{Beh}_1 + \cdots + \text{Beh}_k \mid \text{Beh}_1 \oplus \cdots \oplus \text{Beh}_k$$

where "$\rightarrow$" indicates sequential concatenation operator, "+" inclusive-or operator and "$\oplus$" exclusive-or operator.

Basic interaction element is a four tuples Ele = (PointType, Source, Target, ECADef). Source and Target identify message sender and message receiver; ECADef represents [msgE, Condition, msgA], where *msgE* and *msgA* are message names and *Condition* is Boolean expression.

Modelling e-commerce protocols. The refinement of AUML PD can be used to analyse NetBill protocol. This can be helpful to improve the design of more complex e-commerce protocols. The simplified NetBill protocol is described below.

1. $C \rightarrow M$: goods request (GR)
2. $M \rightarrow C$: encrypted goods with a key K (EG)

3. $C \rightarrow M$: signed electronic Payment Order (EPO)
4. $M \rightarrow B$: endorsed EPO (EEPO) including key K
5. $B \rightarrow M$: payment slip (PS) or no payment (NP)
6. $M \rightarrow C$: PS(inc1uding key K) or NP. If C does not get the receipt for some time, C may inquire B
7. $C \rightarrow B$: transaction enquiry (TE)
8. $B \rightarrow C$: transaction results, PS, NP or no record (NR)

Three participants are involved in the protocol, one consumer, one merchant and one trusted bank. By using the extensions for lifeline and messaging, NetBill can be simplified by RPD. For example, on the lifeline of participants, in-message EEPO and out-message EPO are causally linked; the triggered messaging shows that some internal events cause message passing.

4.2 An ENDL-Based Verification Model

Usually, the modelling of electronic transaction protocols requires distinct specification to transactions. However, the specification of protocols are not always consistent as people expected. This happens very often in the complicated electronic transaction protocols. Thereby, the inconsistent contexts will be considered to be insecure here.

4.2.1 Components

According to different functions, the verification model can be divided into four fundamental modules.

- Inference engine,
- Knowledge base,
- User interface,
- Facts database.

The *knowledge base* comprises the knowledge that is specific to the domain of application, including such things as facts about this domain, rules that describe the relations or phenomena in the domain. The inference rules inside the *knowledge base* imply the fundamental security mechanisms of transaction protocols, which should be kept steady.

Example 4.1. Examples of known facts and rules include *Alice* knows *Tom*'s public signature key, $know(Alice, Spb(Tom))$; and if *Alice* knows communication key k and $e(m, k)$, she should know the plain text of message m.

The *inference engine* is the core of the whole inference framework. It knows how to actively use the knowledge in the knowledge base. This verification

model uses the embedded inference engine of *Prolog*. For simplicity, the discussion of inference engine is removed here. In general, a *user interface* is necessary to provide communication between users and systems. It is convenient to view the inference engine and interface as one module, usually called a *shell*. In addition to the known facts, some real-time facts have to be collected by the user interface, and then stored into the *facts database*.

Example 4.2. Example of real-time facts includes *Alice* knows the book order sent from *Bob*, $know(Alice, book\_order(Bob))$.

From the above description, each of them actually performs as one of the independent function modules of the verification model.

The aforementioned scheme separates knowledge from algorithms that operate on the knowledge. It enables a rational way of developing a verification model for several applications by developing a shell that can be used universally, and then adding some new knowledge for each application. Nevertheless, even if modifications of the knowledge base from one security protocol to another are necessary, the basic security principles should keep the same as before at least. Thus, it is convenient to apply this model to validate various security protocols.

4.2.2 Designing the Model

As mentioned above, there are usually some known facts about encryption keys, messages and some real-time facts. In particular, the known facts are stored in the knowledge base along with the inference rules. However, the real-time facts have to be derived from the interaction between users and verification systems via the *user interface*.

In general, the language of *if-then* rules is adopted for representing knowledge. Each rule actually consists of two parts, condition and conclusion. It is possible that the conditions of a rule is the conclusion of other rules.

Example 4.3. Rule 1, if *Alice* knows communication key k and encrypted message $e(m, k)$ then she should know the plain text of m; *rule* 2, if *Alice* knows message m, she should know the signed message $S(m, Spv(Alice))$. The conclusion of *rule* 1 is actually the condition of *rule* 2.

Thus, the knowledge base can be shown in Figure 4.1 as an inference network. Nodes $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$, $\boldsymbol{G}$, and $\boldsymbol{H}$ in the network correspond to propositions and links correspond to rules in the knowledge base. $\boldsymbol{E}$ and $\boldsymbol{G}$ are the conclusion of rule 2 and rule 3 respectively. However, they are acted as the conditions in rule 4. Therefore, the inference system will firstly search the conclusion from the *facts database* during the verification execution. If it finds the matched facts in the database, the verification process will skip this rule and turn to check the next inference rule for matching the other

conditions; if not, it has to match the conditions of this rule one by one. Arcs that connect some of the links indicate the conjunctive connection between the corresponding propositions. The network representation of Figure 4.1 is in fact an AND/OR graph.

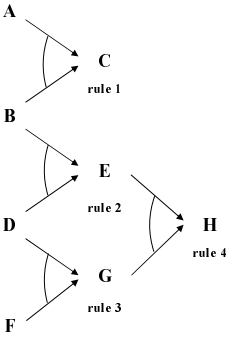

Fig. 4.1. Mode of inference network

Once knowledge is represented in some form, we need a reasoning procedure to draw conclusions from the knowledge base. For if-then rules, there are two basic ways of reasoning [20]:

- backward chaining, and
- forward chaining.

Backwards chaining starts with a hypothesis and works backwards, according to the rules in the knowledge base, toward easily confirmed findings. However, the forward chaining is in the opposite direction. In our verification model, the back chaining is chosen as the reasoning way, which searches from the goal we want to verify to data. Thus, the user needs to submit an authentication command as a goal, and then the verification system will try to search the *facts database* and *knowledge base* to prove the goal. If the verification achieves the goal, the authentication succeeds. In contrast, if the goal cannot be proved, based on existing knowledge, it is natural for us to conclude that the authentication fails due to the *non-monotonic* and *fail-negate* properties of ENDL.

This verification model provides four different ways to handle the information:

- Adopt the external file as the storage of the knowledge base;
- Collect the facts and knowledge via the interaction with user;
- Access the knowledge base;
- Output the results.

The procedures of verification are depicted in Figure 4.2. The verification starts when a user submits an authentication request to the authentication server. As stated above, some known facts should be collected previously and stored into the *knowledge base* together with the inference rules. Also, the user needs to input the real-time facts and store them into the *facts database*. In answer to the authentication request, the authentication server will search the knowledge base in terms of the inference network in Figure 4.1. Suppose the knowledge base includes n inference rules like the set R below:

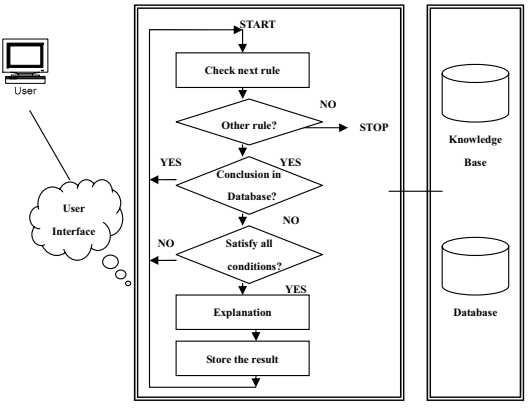

Fig. 4.2. The algorithm flow of inference engine

$$R = \{rule_1,\ rule_2,\ \ldots\ ,rule_n\}$$

where the rules in knowledge base are of the form:

$$rule_i = \{(N,\ [Condition_{ij}],\ Conclusion_i) |\ 1 \leq i \leq \mathrm{n},\ 1 \leq j\}$$

where $Condition_{ij}$ is a set of simple assertions using logic operators *and* and *or*, $Conclusion_i$ is a simple assertion that does not contain logic operator, and N is the rule name. The assertions in rules can be terms that contain variables.

If all rules of R have been searched thoroughly, the authentication will be halted promptly. If a rule $rule_i$ is found, the system will go ahead to check whether the $Conclusion_i$ of this rule has been stored in the *facts database*. If the $Conclusion_i$ is found, the verification system will skip it and go to next rule; if it cannot be found, the system has to match the $Condition_{ij}$ of $rule_i$ one by one. If and only if all the conditions are satisfied, a truth value '*true*' will be returned, and the authentication server will generate a response and store the results into the database; otherwise we have to check next inference rule until all of them are searched out. To match the $Condition_{ij}$, it can be derived from

the fact database or deduced from other inference rules. A response message is then sent to the requester, which is used to decide whether the validated message is secure or not.

To access the knowledge base and fact database, it is necessary to establish an interface between users and systems. This assists in conducting fact collection and query. The details are described below.

4.2.3 Handling the Knowledge and Facts

Two ways can be used to handle the knowledge base of the verification model. One is to input new knowledge, and the other is to read information from the existing knowledge base. They can be denoted by *'a'* and *'b'* respectively.

process ('a') if acquisition.
process('b') if write ('Name of the knowledge base'),
readln (Name),
consult(Name).

As described above, some known facts are public knowledge. For example, *Alice* knows her own private key. Obviously, they will remain true in any security protocols.

Except for the known facts, there are also some real-time facts derived using the interaction with users. Thereby, a predefined functor *facts_reading* is needed to collect the real-time facts. It is triggered promptly once the verification server receives the authentication request from the remote client. Every time, when the user inputs facts, the system will ask users whether it is a terminal symbol "*yes*". If so, the processing of facts collection is halted, otherwise the user will keep on inputting other facts. *Facts_ reading* is in fact a human-interaction facility, in which the user can collect the facts through answering *'who'* and *'what'* questions. For freshness, some facts are temporarily stored into the dynamic buffer of system and will be void automatically once the reasoning ends.

Example 4.4. "*Alice* knows *Bob*'s public signature key" is a collected fact, in which *Alice* corresponds to the *'who'* question and *Bob*'s public signature key *Spb(Bob)* denotes *'what'* question.

In addition, it is very often that the verification system may ask the same question inquired in the last rule to the subgoal of another rule. To avoid repeated questions, the enquired question should be saved in the working storage of system. A built-in procedure *assertz* is used to achieve this goal, which can help the system to memorize the asked question.

The *facts database* is created to save the input facts. Among them, each fact is separated into several fields according to the concrete instances, and is stored into the database. Facts in the database are of the format:

$$Facts = \{(Name_i, Message_i) \mid 0 \leq i\}$$

During the authentication, it is convenient to search the facts from the *facts database* by executing SQL query. Also, the verification system can execute the **INSERT**, **UPDATE**, and **DELETE** operations on facts. This provides a flexible way for the communications among the *inference engine*, *knowledge base*, and *facts database*.

If a user wants to update the knowledge base, he/she can use another built-in procedure *asserta*() to store the new rules into the knowledge base. However, this operation should just be accessed by authorized users under the consideration of security. Thus, the user has to gain the authorization before he starts the process.

4.2.4 Recognition

This section presents how the verification model is used to validate several instances of security protocols, in which the last three instances have been verified in Chapter 3 by using theorem proving. For simplicity, the verification below stresses on *rules handling* and *facts reading*.

Example 4.5. **Merchant certificate request in SET.**

This example is intercepted from the merchant registration in SET. $Auth(CA, M, CertReq)$ is the proposition that we want to verify. $Know(CA, <Kpv(CA), Kpb(CA)>)$, $Know(M, Spb(M))$, and $Know(CA, <Spv(CA), Spb(CA)>)$ are some known facts, which should have been stored into the knowledge base before the verification starts. The real-time facts, such as $Know(CA, Spb(\mathrm{M}))$, will be input by using the *facts_reading* functor. In addition, the messages, $E(<CertReq, S(H(CertReq), Spv(M))>, k_1)$ and $S(<k_1, AcctData(M)>, Kpb(CA))$ etc. sent from M to CA are generated via the user's input. The knowledge is transferred into rules with rule's name, and then stored into the knowledge base. The serial number is used to denote the rule name for brevity.

rule(1, [“$Know(CA, S(<k_1, AcctData(M)>, Kpb(CA)))$”, “$Know(CA, Kpv(CA))$”, “$Know(CA, k_1)$”], “$Know(CA, AcctData(M))$”)
rule(2, [“$Know(CA, E(<CertReq, S(H(CertReq), Spv(M))>, k_1))$”, “$Know(CA, k_1)$”], “$Know(CA, <CertReq, S(H(CertReq), Spv(M))>)$”)
rule(3, [“$Know(CA, <CertReq, S(H(CertReq), Spv(M))>)$”], < “$Know(CA, CertReq)$”, “$Know(CA, S(H(CertReq), Spv(M)))$”>)
rule(4, [“$Know(CA, CertReq)$”, “$Know(CA, S(<ID_M, T, H(CertReq)>, Spv(M)))$”, “$Know(CA, |Clock - T| < \triangle t_1 + \triangle t_2)$”], “$Auth(CA, M, CertReq)$”)

where *Clock* is the local time, $\triangle t_1$ is an interval representing the normal discrepancy between the server's clock and the local clock, and $\triangle t_2$ is an interval representing the expected network delay time [40]. Each rule is separated into

conditions and a conclusion by the square bracket. The rules presented above are assigned with the instance variables, but they are in fact replaced with variables when stored into the knowledge base.

The user then needs to input the real-time facts. They are listed in Table 4.1 with the *Name* and *Message* fields. Each row of the table denotes *who* (Name) knows *message* (Message).

Table 4.1. Merchant Certificate Request.

Principal's name	Message
M	k_1
M	$Kpb(CA)$
M	$CertReq$
CA	$Spb(M)$

After establishing the knowledge base and completing the collection of real-time facts, the user can submit an authentication request:

?- $Auth(CA,\ M,\ CertReq)$.

The verification system then searchs the knowledge base for the matched rules. However, the system cannot authenticate $Auth(CA,\ M,\ CertReq)$ since the merchant did not add the timestamp and identifier in the message [29]. Finally, the verification result is replied to the user. Also, some feasible suggestions will be generated as a reference for future transactions.

Example 4.6. **Purchase response in SET.**

This example is a process of purchase request. $Auth(C,\ M,\ PRes)$ is the proposition that needs to be verified. Firstly, the system collects the facts $Know(C,\ PRes)$ and $Know(C,\ Spb(M))$ by interacting with user. In addition, some known facts, such as $Know(M, <Spb(M),\ Spv(M)>)$, should have been stored in the knowledge base. The fact collection will be conducted until the user inputs a terminal symbol "*yes*". Four inference rules are used in the verification:

rule (1, ["$Know(M,\ PRes)$"], "$Know(M,\ H(PRes))$")
rule (2, ["$Know(M,\ PRes)$", "$Know(M,\ Spv(M))$"], "$Know(M,\ S(PRes,\ Spv(M)))$")
rule (3, ["$Know(M, PRes)$", "$Know(M, Spv(M))$"], "$Know(M, S(H(PRes),\ Spv(M)))$")
rule (4, ["$Know(C, PRes)$", "$Know(C, Spb(M))$", "$Know(C, S(<ID_M,\ T,\ H(PRes)>,\ Spv(M)))$", "$|Clock\text{-}T| < \triangle t_1 + \triangle t_2$"], "$Auth(C,\ M,\ PRes)$")

where the fourth *rule* has been defined in the last example. And thereby, it is unnecessary to generate this rule again.

Table 4.2 describes the storage form of facts, in which the columns of name and message are used to store user name and message, respectively.

The user then submits an authentication command to the authentication server:

?- $Auth(C, M, PRes)$.

Table 4.2. Purchase Response in SET.

Principal's name	Message
M	$PRes$
M	ID_M
C	$Spb(M)$

To match the conditions of a rule, one way is to search the current fact database, and the other is to ask users via the user interface. By doing so, the system fails to authenticate the *PRes* sent by the cardholder. The system stops the authentication and sends a response message to user. At the same time, the system needs to retract the outdated messages, such as outdated facts, encryption keys and transaction ID.

Example 4.7. **Distribution of communication keys in Needham and Schroeder's protocol.**

This instance is the distribution of communication keys in Needham and Schroeder's protocol. It assumes that the authentication server (AS) is responsible for generating and distributing all communication keys, and each user has registered a private(secret) key with the AS.

The key distribution protocol is briefly described below. For a user A to obtain a key CK to share with other user B, three steps are taken:

(1) $A \rightarrow AS$: A, B, ID_A
(2) $AS \rightarrow A$: $\{I_A, B, CK, Y\}^{K_A}$
(3) $A \rightarrow B$: $e(<CK, A>, K_B)$

where K_A is only known to A and AS. After the step (3), A can ensure that CK is safe to use. Then, a handshake between B and A follows:

(4) $B \rightarrow A$: $\{ID_B\}^{CK}$
(5) $A \rightarrow B$: $\{f(ID_B)\}^{CK}$

Table 4.3. Distribution of Communication Keys in Needham's protocol.

Principal's name	Message
A	$<A, B>$
A	ID_A
A	k_A
AS	CK
AS	k_A
AS	$<A, B>$
AS	k_B
B	k_B
B	ID_B

In [40], a replay attack has been reported in this protocol. Here it is validated using the proposed verification model in this chapter. Let $m = e(<CK, A>, K_B)$

$Auth(A, AS, CK)$ is the goal needs to be validated. $Know(A, K_A)$, $Know(AS, K_A)$, $Know(B, K_B)$ and $Know(A, K_B)$ are known facts and have been stored in the knowledge base. The real-time facts include $<A, B, ID_A>$ sent from A to AS, message $e(<ID_A, B, CK, m>, K_A)$ sent from AS to A, and message m sent from A to B. For brevity, the messages transmitted in the last two steps are not included. Table 4.3 describes some real-time facts.

The inference rules used to handle the facts are listed below:

- rule (1, ["$Generate(A, <A, B, ID_A>)$"], "$Know(A, <A, B, ID_A>)$")
- rule (2, ["$Send(A, AS, <A, B, ID_A>)$"], "$Know(AS, <A, B, ID_A>)$")
- rule ($1^{'}$, ["$Generate(AS, <ID_A, B, CK, m>)$"], "$Know(AS, <ID_A, B, CK, m>)$")
- rule ($2^{'}$, ["$Send(AS, A, <ID_A, B, CK, m>)$"], "$Know(A, <ID_A, B, CK, m>)$")
- rule (3, ["$Know(AS, k_A)$", "$Know(AS, <ID_A, B, CK, m>)$"], "$Know(AS, e(<ID_A, B, CK, m>, k_A))$")
- rule ($3^{'}$, ["$Know(A, k_B)$", "$Know(A, <CK, A>)$"], "$Know(A, m)$")
- rule ($2^{''}$, ["$Send(A, B, m)$"], "$Know(B, m)$")
- rule (4, ["$Know(A, k_A)$", "$Know(A, e(<B, CK, T, m>, k_A))$", "$|Clock\text{-}T| < \triangle t_1 + \triangle t_2$"], "$Auth(A, AS, CK)$")
- rule ($4^{'}$, ["$Know(B, k_B)$", "$Know(B, e(<A, CK, T>, k_B))$", "$|Clock\text{-}T| < \triangle t_1 + \triangle t_2$"], "$Auth(B, A, CK)$")

The rule (1) and ($1^{'}$), rule (2), ($2^{'}$) and ($2^{''}$), rule (3), ($3^{'}$), and rule (4) and ($4^{'}$) are actually regarded as four rules in knowledge bases. Among them, rule (4) is generated in light of the proved result in [40]. The variables of rules will be initiated with instance variables during validation.

The user then sends two authentication commands to the authentication server:

?- $Auth(A, AS, CK)$.
?- $Auth(B, A, CK)$.

The system fails to validate CK for the same reason mentioned in [40]. The system then responds an error message to the user.

Example 4.8. **Distribution of Public keys in Needham and Schroeder's protocol.**

In this example, A intends to attain B's public key by consulting with the authentication server AS. In addition, the B's public key must be verified to guarantee its validity. Here the authentication server (AS) is responsible for storing and distributing users' public keys.

Table 4.4. Distribution of Communication Keys in Needham's protocol.

Principal's name	Message
A	$<A, B>$
AS	$<A, B>$
AS	CK
A	ID_A
B	ID_B

To validate the goal $Auth(A, AS, Spb(B))$, the verification system may usually handle known facts, real-time facts and inference rules. The known facts consist of $Know(A, Spb(AS))$, $Know(A, Spb(B))$, $Know(B, Spb(A))$ and $Know(AS, Spv(AS))$, which have been stored in the knowledge base previously. In addition, the real-time facts include $<A, B>$ generated by A and known by AS, $CK = <B, Spb(B)>$(communication key) known by AS, ID_A known by A and ID_B known by B. All the real-time facts will be stored as the form in Table 4.4.

To transmit messages between A, B and AS safely, the transmission usually conforms to the inference rules to implement encryption, signature, decryption, and authentication. For convenience, the variables of inference rules are assigned instance variables.

rule (1, ["$Generate(A, <A, B>)$"], "$Know(A, <A, B>)$")
rule (2, ["$Send(A, AS, <A, B>)$"], "$Know(AS, <A, B>)$")
rule (3, ["$Know(AS, CK)$", "$Know(AS, Spv(AS))$"], "$Know(AS, S(CK, Spv(AS)))$")

rule $(1^{'}$, [“$Generate(AS, S(CK,Spv(AS)))$”], “$Know(AS, S(CK, Spv(AS)))$”)
rule $(2^{'}$, [“$Send(AS, A, S(CK, Spv(AS)))$”], “$Know(A, S(CK, Spv(AS)))$”)
rule (4, [“$Know(A, Spb(AS))$”, “$Know(A, S(<T, B, Spb(B)>, Spv(AS)))$”), “$|Clock\text{-}T| < \triangle t_1 + \triangle t_2$”], “$Auth(A, AS, Spb(B))$”)

The verification begins when user sends an authentication command to the authentication sever:

?- $Auth(A, AS, Spb(B))$.

The goal $Auth(A, AS, Spb(B))$ cannot be verified since the conditions in rule 4 are not satisfied. It actually obtains the same result as in Chapter 3 by using theorem proving.

Example 4.9. **Initial response in Cardholder Registration.**

As described in Chapter 3, the cardholder C and certificate authority CA have to authenticate the received messages. $Auth(C, CA, InitRes)$, $Auth(C, CA, <Spv(CA), Spb(CA)>)$ and $Auth(C, CA, <Kpv(CA), Kpb(CA)>)$ are the goals that we want to validate.

Table 4.5. Initial response in Cardholder Registration.

Principal's name	Message
C	$RegFormReq$
C	PAN
C	k
CA	$InitRes$

The known facts in this instance are $Kpb(CA)$, $Spb(CA)$ known by C, and $CertS(CA)$, $CertK(CA)$ and $Spv(CA)$. In addition, it includes some real-time facts during the transaction, including k, $RegFormReq$ and PAN generated by C, and $InitRes$ generated by CA, which are depicted in Table 4.5.

The following rules are used to encrypt and decrypt the transmitted messages by C and CA.

rule (1, [“$Generate(X, InitRes)$”], “$Know(X, InitRes)$”)
rule (2, [“$Send(X, Y, InitRes)$”], “$Know(Y, InitRes)$”)
rule (3, [“$Know(X, Spb(CA))$”, “$Know(X, S(Message, Spv(CA)))$”], “$Know(X, Message)$”)
rule (4, [“$Know(X, InitRes)$”], “$Know(X, H(InitRes))$”)
rule (5, [“$Know(X, CertS(CA)$”, “$Verify(X, CertS(CA), CA)$”], “$Auth(X, CA, <Spb(CA), Spv(CA)>)$”)

rule (6, [“$Know(X, CertK(CA)$”, “$Verify(X, CertK(CA), CA)$”], “$Auth(X, CA, <Kpb(CA), Kpv(CA)>)$”)
rule (7, [“$Know(X, Spb(CA))$”, “$Know(X, InitRes)$”, “$Know(X, S(<ID_Y, T, H(InitRes)>, Spv(CA)))$”, “$|Clock\text{-}T| < \triangle t_1 + \triangle t_2$”], “$Auth(X, CA, InitRes)$”)

Among them, the seventh rule has been actually generated in *Example 4.8*.

The verification starts when the user sends a verification command to the verification system.

?-$Auth(C, CA, InitRes)$

The system validates it by handling the facts and knowledge base in light of the process in Figure 4.2. It fails to verify this goal since the *InitRes* is suspected to be subject to replay attacks.

Example 4.10. **Certificate request in Cardholder Registration.**

This example is also a process of Cardholder Registration. It intends to validate the certificate request message sent from *CA* to *C*.

It is unnecessary for the user to input the known facts, such as $Know(C, Spb(CA))$ and $Know(C, Spb(C))$, since they have been stored previously. Then, the main task is to gather the real-time facts, which include k_2, k_3, *CertReq* and *AcctInf* generated by cardholder, and RegForm generated by *CA*. They are stored as the form of Table 4.6.

Table 4.6. Certificate request in Cardholder Registration.

Principal's name	Message
C	k_2
C	k_3
C	*CertReq*
C	*AcctInf*
CA	*RegForm*

The inference rules to verify this instance are listed below. Let $m = <CertReq, k_2, Spb(C)>$.

rule (1, [“$Generate(C, <Spb(C), Spv(C)>)$”], “$Know(C, <Spb(C), Spv(C)>)$”)
rule (1′, [“$Generate(C, CertReq)$”], “$Know(C, CertReq)$”)
rule (1″, [“$Generate(C, <k_2, k_3>)$”], “$Know(C, <k_2, k_3>)$”)
rule (2, [“$Know(C, m)$”], “$Know(C, H(m))$”)
rule (3, [“$Know(C, H(m))$”, “$Know(C, Spv(C))$”], “$Know(C, S(H(m), Spv(C)))$”)

rule (4, [$Know(C, <m, S(H(m), Spv(C))>)$, "$Know(C, k_3)$"], "$Know(C, e(<m, S(H(m), Spv(C))>, k_3))$")
rule (5 [$Know(C, <k_3,, AcctInf>)$, "$Know(C, Kpb(CA))$"], "$Know(C, E(<k_3, AcctInf>, Kpb(CA)))$")
rule (6, ["$Send(C, CA, e(<m, S(H(m), Spv(C))>, k_3))$"], "$Know(CA, e(<m, S(H(m), Spv(C))>, k_3))$")
rule (6′, ["$Send(C, CA, E(<k_3, AcctInf>))$"], "$Know(CA, E(<k_3, AcctInf>, Kpb(CA)))$")
rule (7, ["$Know(CA, E(<k_3, AcctInf>))$", "$Know(CA, Kpv(CA))$"], "$Know(CA, k_3)$")
rule (8, ["$Know(CA, e(<m, S(H(m), Spv(C))>, k_3))$", "$Know(CA, k_3)$"] "$Know(CA, S(H(m), Spv(C)))$")
rule (9, ["$Know(CA, S(<ID_C, T, H(m)>, Spv(C))>)$", "$Know(CA, Spb(C))$", "$|Clock\text{-}T| < \triangle t_1 + \triangle t_2$"], "$Auth(CA, C, m)$")

The verification fails to validate the validity of message m transmitted from C to CA due to the same reason as the last example.

4.3 Comparison with Theorem Proving

This section presents a brief comparison between the verification model and theorem proving, and highlights some features of verification model.

Speediness. The verification model usually brings on faster validation than theorem proving. In general, the verification of protocols consists of three steps:

- Collecting facts, including known facts and real-time facts;
- Submitting the authentication goal;
- Searching the matched rules.

Table 4.7. Speediness of Verification Model

	Theorem Proving	Verification Model
known facts	not recorded	recorded and automatically activated next time
real-time facts	input manually	input via user interface
rules	matched manually	searched automatically from knowledge base

It is very often that the theorem proving has to go through all the steps, which are difficult to be implemented manually and time consuming. A user has to collect some repeated facts again in next validation for they were not stored

into knowledge base. Table 4.7 gives a simple contrast between verification model and theorem proving from three aspects.

From the description in Table 4.7, the verification model avoids the repeated input owing to some common knowledge, and realizes the automatic search of inference rules. And thereby, it is usually faster than theorem proving.

Extensibility. As mentioned before, ENDL (a theorem proving method) is extensible, and has been used to formally analyse some examples of security protocols. However, the extensibility of theorem proving is limited since the extended rules cannot be recorded and used in validation next time. Their differences are presented in Table 4.8 from two aspects.

Table 4.8. Extensibility of Verification Model

	Theorem Proving	Verification Model
real-time facts	collected before verification	input randomly
rules	generated before verification	added randomly

In Table 4.8, both the facts collection and rules generation can be randomly implemented in verification model without the limitation of time like theorem proving.

On the other hand, one can look back the *Non-monotonic rule* in chapter 3. This rule is no longer able to describe the verification appropriately since the action α can be altered randomly in the verification model. Let K, F and G be knowledge base, the set of known facts and authentication goal respectively. Based on the rule, we can define that the authentication fails if the verification model cannot find the matched rules after searching the knowledge base and facts thoroughly. The verification is formalized as follows:

Theorem 4.1. *Let X, Y be the message sender and receiver respectively. Let* S_0 *be a set of knowledge. Suppose* $S = K \cup F$. *The authentication then can be defined as a formula:*

$$Auth(X, Y, G) = \begin{cases} True\ if\ \exists S_0 \subseteq S,\ S_0 \neq \phi,\ then\ S_0 \Rightarrow G \\ False\ otherwise \end{cases}$$

This result is another answer to the non-monotonic rule. The verification is possible, in a sense, to go continuously from one rule to the other until all rules have been exhausted.

4.4 Discussion

There have been many other model checkers to protocol verification. Some specification verification tools are powerful but usually less flexible. Thus, it is not easy to definitely determine which one is more appropriate than the others. It depends on the user's objectives and the complexity of the protocols. In the followings, we present several verification tools that have been widely used for protocol analysis.

To verify security protocols in a systematic way, significant efforts have been conducted to develop formal analysis of cryptographic protocols, and many good models have been proposed. Generally speaking, these methods can be broadly classified into two categories, namely state based methods and rule based methods.

The former methods model security protocols using finite state machines. They search the state space exhaustively to see whether all the reachable states are safe [129]. If some reachable state in a security protocol is proved to be unsafe, a flaw may be identified; otherwise, the protocol will be reported correct and safe. In contrast to rule based methods, they are usually complete and can identify most flaws in protocols.

Unlike state based methods, the latter formally expresses what principals can infer from the messages received [129]. The protocols, the needed assumptions and the goals of the protocols are formalized in traditional formal logic. Furthermore, some specific properties of the protocols can be proved by using the axioms and rules of the logic. Rule based methods are not subject to large state space, and can normally run quickly.

To gain effectiveness from state based methods and efficiency from rule based methods, a new security protocol verification method, which is based on a knowledge-based framework and implemented in a mechanical reasoning platform, Isabelle is proposed in [101]. The method analyses the knowledge of participating principals and infers what they can know and can never know. It takes protocols concerning multiple interleaving sessions into consideration and can find flaws which are often overlooked by many rule based methods.

Isabelle is a popular generic theorem prover developed at Cambridge University (Larry Paulson) and TU Munich (Tobias Nipkow). Isabelle/HOL [124] is an interactive proof tool for higher-order logic. Isabelle provides excellent notational support: allows for the introduction of new notation, use of normal mathematical symbols, and natural deduction and sequent calculi. New logics can be introduced by specifying their syntax and rules of inference. Proofs can be written in a traditional proof style or straightforwardly as sequences of commands. It comes with a large theory library of formally verified mathematics, and offers a simple proof formulation language to enable a user to write comprehensible proof scripts. Isabelle strongly supports inductive definitions, both in its specification language and in its prover automation.

Isabelle has many features in common with HOL, whereas, unlike HOL, Isabelle provides direct support for unification. Unification underlies Isabelle's predicate calculus proof procedures, which can prove quite complicated formulae. Much recent work in HOL involves embedded logics. The semantics of each symbol is expressed in higher-order logic to embed a logic. The parser generator of Isabelle enables easy definition of the syntax and construction of rules without programming. This makes it particularly well suited for supporting embedded logics.

Although Isabelle can cope with arbitrary large systems, even infinite in the number of states, a disadvantage is the necessary interaction of the user during verification process. On the other hand, model checkers enable automatical verification, whereas their use is usually restricted to small and finite systems.

FDR [53] is a CSP model checker. It privides the choice of verification using any of the three models of CSP: *Traces Refinement*, *Failures Refinement*, and *Failures-Divergences Refinement*. FDR provides a graphical interface for determining the source of errors by analysing the trace of events that led up to the error. There are some features of security protocols which make the verification well-suited to CSP. First, CSP is appropriate for modelling multi-agent systems which exchange messages with each other. CSP has close attention to issues such as insecure message transport in a protocol. CSP can model the insecure process over series of actions as the possible behaviours it can perform rather than a single action or individual state. In addition, it is feasible to include time into CSP modelling, either continuous time (in abstract models) or discrete (for model checking). These properties seem to be well-related to use of time and time-stamp in security protocols.

SPIN [58] is a generic verification system that supports the design and verification of asynchronous process systems. SPIN verification models aim to prove the correctness of process interactions. Unlike other approaches to model checking, SPIN is focused on asynchronous control rather than synchronous control. To sum up, SPIN aims to provide an intuitive notation for design specification written in the verification language PROMELA, a concise notation for expressing general correctness requirements in the syntax of standard Linear Temporal Logic (LTL).

SPIN has its application in temporal logical model checking and in the protocol verification systems based on on-the-fly reachability analysis. It has been applied to perform verification run to prove some basic safety properties, such as absence of deadlock, unreachable code and unspecified receptions, and prove liveness properties, such as faithful message transfer. Recently, SPIN has also been applied to the verification of security protocols [103]. The purpose of applying SPIN to security protocols is to verify their resistance to malicious manipulation, which is usually not required in normal communication protocols.

It is observed that the above model checking methods show different capabilities in different applications under various situations. To the best of our knowledge, it is not easy to realize fully automatical verification. The selection of methods may have to depend on the complexity and the types of protocols.

4.5 Summary

The theorem proving has been widely used to validate the key exchange and authentication protocols in the traditional methods mentioned in Chapter 1. However, not much work has been found in the model checking of electronic transaction protocols.

The correctness conditions for electronic transaction protocol usually contain more components than the usual security protocols. Moreover, the number of messages and participants of electronic transaction protocols is difficult to bind.

Recently, there are considerable efforts to develop methods for formal analysis of e-commerce protocols by using model checking. They are concerned about different specific aspects and usually use different languages or tools. For example, the failure analysis of e-commerce protocols proposed by [76], and an automatic analysis of e-commerce protocols proposed by [157] using the Unified Modeling Language (UML). They are usually too difficult to use due to the complexity of used languages or tools.

To address these problems, this chapter proposes a verification model based on ENDL especially used for the validation of electronic transaction protocols. We can write precise definitions of the behaviour of a protocol (at any desired level of abstraction), formulate protocol properties and test that they are satisfied via this model. Moreover, the user interface in this model provides user with a convenient way to collect the real-time facts and new knowledge. These policies assist in overcoming the low efficiency and error prone in theorem proving. In particular, the properties of speediness and extensibility make it a good complement to the conventional theorem proving.

Some instances of SET and Needham protocols are validated by using this model. It proves that the model checking is useful and effective in analysing security protocols.

5

Uncertainty Issues in Secure Messages

Traditional protocol analysis ideally assumes that the transmitted messages are secure and consistent between principals. However, the described hostile environments might cause inconsistent messages such as intercepted, missing, or tampered messages, and conflicting beliefs in the messages. It is very difficult to express these situations using previous methods. Although the inconsistency of secure messages and the degree of conflicting beliefs can aid in evaluating the performance of protocols, they are usually uncertain rather than simply true or false. As important complement to protocol analysis, this chapter discusses two primary uncertainty issues including the estimation of inconsistency of messages and the integration of conflicting beliefs. We not only present intuitive ways to measure both uncertainties but also give some case studies to illustrate the usefulness of our approaches.

5.1 Introduction

Electronic commerce (e-commerce) has been emerging as a new business mode, lodging with the electronic funds transfer (EFT), electronic data interchange (EDI), and the World Wide Web (or Internet). The emergence of e-commerce has innovated upon the current business practice, and broken through conventional marketing barriers, as activities on the Internet are no longer limited to time and geography. Thereby, e-commerce presents unprecedented potential on cost reduction and the convenience of businesses, as well as brings new challenges. This has leaded to the development of a great many e-commerce systems. For example, (1) governments, hospitals and hotels have purchased their supplies on the Internet, and (2) companies like **Cisco** and **Dell** have been successful in creating electronic storefronts for selling.

With the wide application of e-commerce systems, a number of very serious limitations to this work have been arisen in part from the fact that purchasing goods or services in e-commerce systems are transmitted through the open

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 131–173, 2008.

network that can suffer from the malicious attacks. For example, simply delivering the messages such as credit card number on the Internet leaves them open to fraud. This has led largely to the development of security measures. To succeed in the fiercely competitive e-commerce marketplace, people must become fully aware of security threats, utilize the technology that overcomes them, and win customer's trust.

In general, security in e-commerce is implemented by relying on a set of security protocols that meet the user's expectation for secure transactions. A good security protocol must be able to state not only the overall objectives of transactions in terms of integrity and confidentiality, but also the range of circumstances under which these objectives must be met, otherwise there may exist the possibility of attacks on the transmitted messages. For example, users may suspect a transmitted message to be insecure for the incomplete or ambiguous specification of protocols. Consequently, security protocols have become the requisite of e-commerce systems. And many methods and tools have been developed to verify these protocols as described in Chapter 1. However, their applications are relatively mechanical in e-commerce systems.

A variety of formal methods, representing the belief and/or knowledge have been developed to analyse the protocols [22]. They are either concerned with measuring the trust that can be put on the goal by the legitimate communicants using beliefs of the principals, or analysing the security of a protocol by examining the knowledge gained by an intruder in the process of the protocol [24]. Usually, the analysis in the first group starts from formalizing a set of exchanged messages, applying the inference rules, and then deducing the stated goal of the protocol.

Nevertheless, there is a lack of true intellectual process by using the inconsistency removing. This is demonstrated by the fact that the messages transmitted between principals can be inconsistent owing to the potential communication block, message lost and/or malicious attacks. Also, in a malicious environment, the belief of principals in transmitted messages can no longer be justified. And the current formal methods to verify security protocols usually assume that the principals are trustworthy and the communication channels are secure, and fail to model insecurity. Thus, it is critical to have the capability of modelling the imperfect working conditions and verifying the protocol in such circumstances. Unfortunately, no much effort has been found to handle the aforementioned problems owing to the difficulty in modelling the uncertain and insecure situations. This prevents us from evaluating the performance of the protocol.

A variety of forms of Internet security problems have been reported, such as viruses, interception and fraud. They can result in the inconsistency issues mentioned above. One solution is to resend all messages, whereas this is not an intelligent behaviour. Another solution is to use the modern cryptographic algorithms such as the block cipher [130] to avoid the conflicting messages.

This signifies the potential risks that we may run into. Therefore, current techniques have not touched on the topic of measuring the inconsistency in secure messages.

There have been considerable efforts to deal with the inconsistency and uncertainty in knowledge base. Usually, they focus on either merging inconsistent knowledge [93, 95, 96] or measuring inconsistency between knowledge bases [71], or evaluating probability based belief [51]. However, they rarely talk about dealing with inconsistent messages and conflicting beliefs in messages. Unlike general knowledge, the characteristics of secure messages need to be considered.

Compared with the general knowledge, secure messages have some distinct properties, such as freshness and dynamics. More specifically, before handling the uncertainty in secure messages, we must not only ensure that they are not reply attacks but also confirm the messages are really derived from the sender and received by whom he claims to be. In addition, the principals of secure messages may be associated with a weight presenting the degree of importance, such as the hierarchy of trust in *Public Key Infrastructure* (PKI) in Figure 3.2. Moreover, the belief relationship for disjunction connectives in knowledge base cannot be applied in the secure messages for the reason that the principal should not allow to have ambiguous opinion about the support of a secure message. For example, let α and β be two secure messages. It is unallowable for the principal P to support $\alpha \vee \beta$ but he must support either α or β. All these distinct features provide a significant insight into the difficulty of solving the uncertainty in secure messages.

This chapter first present a formal logical framework [31] designed for estimating the inconsistency in secure messages. The approach is based on the weighting majority and takes into account the freshness and dynamics of secure messages. It achieves (1) measuring the inconsistency in secure messages with weights that represent the degree of importance of principals; and (2) analysing the inconsistent secure messages by numerically measuring their reliability. We also proposes method [33] to handle the inconsistent belief between principals. It allows probabilistic rules by combining probabilities with the rule. We propose a probabilistic semantic for ENDL [29], in which the probabilities of sentences and rules are viewed as observed belief and inferred belief, respectively, and can be modelled in various levels of detail. A numeric estimation to the partial belief is presented by computing the minimum trust in the authenticated goal of the protocol. The examples and experiments demonstrate that the designed frameworks are effective enhancing existing formal analysis of security protocols.

The rest of this chapter is organized as follows. Section 5.1 gives a brief introduction to background knowledge. In Section 5.2, we present the framework to estimate the inconsistency of secure messages, including experiments using a cash withdraw transaction from ATM. Section 5.3 shows the inte-

gration of conflicting beliefs in secure messages, and illustrate it using two experiments. Finally, we conclude our contributions in Section 5.4.

5.2 Estimation of Inconsistency of Secure Messages

Messages that are transmitted between senders, receivers and the third party may be inconsistent due to potential communication block, message lost and/or malicious attacks, in an electronic transaction. And current formal methods to verify security protocols are unable to deal with the inconsistent secure messages between different principals or at different moments. Ideally, they assume that the participants of the protocol are trustworthy and the communication channels are reliable. This may lead to e-commerce activities at risk. In this chapter, a formal framework is proposed to handle the inconsistency in secure messages using weighting majority, which takes into account the properties of secure messages. It measures intuitively the inconsistency in secure messages by providing a numeric estimation, and classifies the secure messages into three categories by using the results. This enables us to formally verify security protocols under hostile and/or uncertain communication environments.

5.2.1 Related Work

Information Security in E-Commerce. In general, the business messages in e-commerce should be electronically transmitted in some manners, and therefore, security services are required to ensure reliable, trustworthy electronic transmission of the messages. The security mechanisms are usually classified into three categories [48, 59, 102] as follows:

- System security means nothing happens to the computer and equipment, including virus, logic bomb etc.
- Network security means only the authorized users can access the network and do what he/she can do under the privilege assigned to him/her.
- Data security means how to ensure the integrity and confidentiality of transmission message.

The primary method to achieve data security is encryption [13, 23] that is a process of encoding a message so that the meaning of the message is not obvious. The reverse process is called decryption by transforming an encrypted message back into its normal form.

The basic methods to achieve the goal of data security are as follows.

- **Symmetric cryptography** where entities share a common secret key depicted in Figure 2.2. In symmetric key systems, the primary encryption algorithm in use is the Data Encryption Standard (DES) [153].

- **Public key cryptography**, namely asymmetric cryptography, where each communicating entity has a unique key pair, a public key and a private key depicted in Figure 2.3. When one key of a key pair is used to encrypt a message, the other key from that pair is required to decrypt the message. If the private key is used to sign a message, the public key from that pair must be used to validate the signature. It was introduced by Whitfield Diffie and Martin Hellman [42]. Diffie-Hellman cryptography is still used today, but it has some vulnerability to a man-in-the-middle attack. A popular algorithm used for key pair generation is RSA (Rivest-Shamir-Adelman) algorithm [132]. Each user generally has two public/private key pairs. One key pair is used to encrypt session keys and the other to create digital signatures. These are known as the key exchange key pair and the signature key pair, respectively.
- **Certificate authority** is important players in authentication, message integrity and non-monotonic security measures. Public-private key pairs are normally registered with a trusted certification authority.
- **Digital signature** is designed to bind the message originator with the exact contents of the message. The sender uses his/her private key to compute the digital signature. In order to calculate the digital signature, a one-way hashing algorithm (such as SHA-1 [134]) may be used to firstly calculate a message digest. The sender's private key is used at this point to encrypt the message digest. The encrypted message digest is what is commonly referred to as digital signature.

Except for the above methods, the **access control**, such as *password*, *fingerprint* [161], and *smartcard* [80] can help to identify authorized users and provide authentication. In addition, another powerful method is *firewall* [104]. Erecting a firewall between sensitive transactions information and untrustworthy networks is essential to prevent security break-ins through vulnerabilities in the operating system.

These security services have been deployed to relieve the security concerns. Among them, data security of transactions is usually highlighted. Without the data security, information transmitted over the Internet is susceptible to fraud and other misuses. Hence the security of electronic commerce has to be recognized when doing transactions over the open network.

Research into Security Protocols. Security measures are usually integrated into the security protocols, which are agreement upon methods of communicating and transmitting data between telecommunication devices. A number of security protocols, such as *communication protocols*, *authentication protocols* and *secure transaction protocols*, have been developed to achieve the security objectives of different layers. Figure 2.1 depicts the function of security protocols. Some of the protocols are as follows.

- *Transmission Control Protocol/Internet Protocol (TCP/IP)* [64] is a widely used protocol on the Internet. TCP conducts at the transport layer of the Open System Interconnection (OSI) model, while IP operates at the network layer. The transport layer provides data reliability and integrity checks of the data received. Network layer performs the data routing and delivery.
- The protocol that underlies the WWW is called the HyperText Transfer/Transport Protocol (HTTP) [151], which handles on the top of TCP protocol. Its primary purpose is to define message formats, message transmissions, and web server and browser commands.
- In order to deal with the security concerns, the *SSL (Secure Socket Layer)* [54] is responsible for routing messages across networks from their source to their destination. SSL adds security inserting itself between the HTTP application and TCP.
- *Secure Electronic Transaction (SET) protocol* was developed with the goal of providing a secure payment environment for the transmission of credit card data.

In an unstable and even insecure environment where e-commerce systems are conducted, security in e-commerce systems is implemented by depending on a set of security protocols. They are responsible for generating, delivering, encrypting and authenticating secure messages, and achieving the trustworthy transmission of messages in distributed environment. A good security protocol must declare not only the objectives of each transaction regarding integrity and confidentiality but also the range of circumstances under which they must be met. However, the security protocols easily suffer from malicious attacks and their designs are a difficult and error-prone task [65].

A variety of methods and tools have been developed to formally analyse the protocols [97, 120]. This has gained many attentions in recent years, such as the work in [22, 44, 62]. In [65], they are mainly classified into two categorizes: those constructing possible attacks using algebraic properties of the algorithms in the protocols (called attack-construction approaches), and those designing inferences using specialized logics based on a notion of knowledge and 'belief' (called inference-construction approaches).

The verification starts from collecting the authentication messages exchanged between principals. It ideally assumes that the communication channels and principals are secure and trustworthy. However, the messages are often inconsistent between principals owing to the hostile/uncertain environment. Unfortunately, the current formal analysis of security protocols are lacking in handling the inconsistent messages.

The modern cryptographic algorithms, such as the public key algorithms [42], secret key algorithms [153] and block cipher [130], have greatly reduced the inconsistency between principals, whereas there are still latent risks that are not easily apparent even to careful inspector. For example, the

sender and receiver may record items with different values possibly due to system error, message lost or broken cipher. Therefore, such inconsistency is still a big challenge to us.

Research into Inconsistency. The above problem can be viewed as a general problem of measuring the inconsistency of secure messages from different principals or at different moments. However, current techniques focus on dealing with inconsistency in knowledge base rather than secure messages. There are many approaches in tackling the inconsistency in knowledge bases, such as, arbitration based information merging [93] and majority based information merging [96]. They are a good foundation to deal with the inconsistency in secure messages.

Shaerf and Cadoli [135] proposed the approximating entailment, in which two sequences of entailment relationship are defined. In [93], Liberatore gave a merging process arbitration to merge the different views between different sources of information and investigated the properties any arbitration operator should satisfy. It intends to preserve as much information as possible after merging; Lin defined a merging operator by majority in contrast with arbitration to merge knowledge base in [96], and gave formal semantics to merge multiple knowledge bases with weight using the method of Dalal [39] in [95]; Konieczny considered the problem of merging several belief bases in the presence of integrity constrains [89], and explored the frontier between merging operator and arbitration operator in [90]; the epistemic entrenchment is an ordering over formulae that indicates the preference for which formulae to give up in case of inconsistency [56]. However, none of them discusses measuring the inconsistency.

Hunter then proposed a method to measure inconsistency in knowledge via QC model [71], and presented a framework to evaluate the significance of inconsistencies [72]. However, the above methods put their emphasis upon measuring the inconsistency of knowledge rather than secure messages. In addition, Fagin and Halpern [51], and Campbell et al. [24] have been successful in dealing with the uncertainty by probabilistic methods but focusing on the belief of principals.

Although the inconsistency in secure messages has been a big challenge to the reliability of verification results, no much work has been found in this filed. In addition, secure messages include distinct properties such as freshness and dynamics, and every message source may be associated with a weight. This urges us to develop new methods and techniques to solve such issues.

5.2.2 Semantics Description

This section simply describes the arisen issues and then presents some basic concepts, symbols and formal semantics.

Problem Description. Like existing verification techniques, we assume that the secure messages are secure during delivery. However, some secure messages have been reported with contradictory values as mentioned above. In other words, this may cause inconsistency.

A logic with three values is adopted in this chapter. In addition to the truth values t and f, an intermediate truth value $u(uncertain)$ is introduced. The truth value u virtually indicates an intermediate level of belief between *true* (i.e., believe) and *false* (i.e., disbelieve).

This three-valued logic is chosen because it provides useful ways to represent the belief in transmitted messages.

1) messages in the first category are definitely insecure to the transaction;
2) whereas messages in the second are surely reliable; and
3) the third cluster is the most important, in which messages are uncertain to be secure or not.

where the third one prevents us from evaluating the reliability of a transaction. We have to combine it with the former two categories and provide an intuitive way to evaluate the inconsistency synthetically .

Moreover, the transmission of secure messages has dynamics properties. They usually go through the procedure of generating, sending and receiving among principals. Furthermore, the freshness is a prerequisite of secure messages to ensure their validity, which is usually realized by timestamp. This is in contrast with existing approaches such as [71, 95] where they do not take into account the dynamics and freshness properties of secure messages. In addition, the notion of entailment is used to depict the causality of a secure message, such as generating and sending, sending and receiving of a secure message. For brevity, the integrity and confidentiality of secure messages are not considered in this framework.

Example 5.1. Suppose $\phi = X$ *knows* $m \wedge X$ *knows symmetric key* k, $\varphi = X$ *knows* $e(m, k)$, φ is hence entailed by $\{\phi \rightarrow \varphi, \phi\}$. Actually, it can be described as $(\phi \rightarrow \varphi) \wedge \phi \Rightarrow \varphi$.

On the other hand, it is possible that the views of some principals are regarded as being more important than the views of the others, such as the different certificate authorities in PKI tree of Figure 3.2. They are usually assigned with weight to present their degree of importance. It is obvious that *CARoot* has the highest authority in the PKI tree.

As already mentioned the existing methods focus on merging knowledge base and do not provide a way to handle the inconsistency in secure messages. This chapter hence proposes a formal framework to deal with the inconsistency in secure messages with weight, in which it takes into account the dynamics and freshness properties of secure messages. It intends to ensure the reliability of verification results of security protocols.

Basic Symbols and Formal Semantics. Suppose $\mathcal{L}$ denotes a set of proposition formulae formed in the usual way from a set of atom symbols $\mathcal{A}$. In particular, $\mathcal{A}$ can contain α and $\neg\alpha$ for some atom α. The logical operators $\wedge$, $\vee$, $\neg$ and $\rightarrow$ denote the connectives. We use variables X, Y, P and CA for principals, Greek letters φ, ϕ and ψ for formulae, $T_{expiration}$ for expiration time of message, T for timestamp, and m, α, γ, θ, μ and $\beta \in \mathcal{A}$ for messages in general. Let $\equiv$ be logical equivalence. A model of a formula ϕ is a possible set of atoms where ϕ is true in the usual sense. ϖ is the weight of message sources. Let k be a key.

On the other hand, there are some operations on a message, which consist of encryption, signature and message digest.

- $e(m, k)$ represents the message m is encrypted by the symmetric key k;
- $S(m, k)$ represents the signed message m by the private signature key k;
- $E(m, k)$ represents the message m is encrypted by the public key-exchange key k;
- $H(m)$ represents the hashing of message m.

In particular, when the message digest of a message is encrypted using a sender's private key, and is appended to the original message, the result is then known as the digital signature of the message. The above function words are the infrastructure necessary to describe the complicated cryptographic operations. For brevity, the technique details are ignored here since we focus on dealing with the inconsistency in secure messages. In addition, we have the following operators:

- <- , -> :: $Message_1 \times Message_2 \longrightarrow Message$, which denotes a set of messages. Moreover, each of them can be a combination of several messages.
- - *sends* - , - :: $Principal_1 \times Principal_2 \times Message \longrightarrow Formula$, which denotes the message was transmitted from $Principal_1$ to $Principal_2$.
- - *generates* -:: $Principal \times Message \longrightarrow Formula$, which denotes the message is generated by *Principal*.
- - *knows* -:: $Principal \times Message \longrightarrow Formula$, which denotes the message is known by *Principal*.
- - *sees* -:: $Principal \times Message \longrightarrow Formula$, which denotes the message has been received by *Principal*.
- *fresh* :: $Message \Rightarrow Formula$, which denotes the message is not a replay message.
- - *believes* - , - :: $Principal_1 \times Principal_2 \times Message \longrightarrow Formula$, which denotes $Principal_1$ believes the message is fresh, and really from $Principal_2$.

Example 5.2. Suppose m, m_1 and m_2 are messages and P_1 and P_2 are principals. $<m_1, m_2>$ denotes a combination of messages; 'P_1 sends P_2, m" denotes the message m is sent from P_1 to P_2; 'P_1 generates m" denotes m is generated by P_1; 'P_1 knows m" denotes m is known by P_1; 'P_2 sees m" represents

principal P_2 has received message m; 'fresh m" represents m is fresh and not a replay of the previous messages; 'P_2 believes P_1, m" denotes P_2 believes the message m is fresh and really from P_1.

In general, facts are stated in the form of expressions called *sentences*, or sometimes well-formed formulae. We define an *atomic sentence* is formed from an n-ary relation operator π mentioned above and n atom symbols a_1, a_2, ..., a_n, by combining them as follows.

$$\pi\ (a_1,\ a_2,\ \ldots,\ a_n)$$

where the atom symbols can be principals, such as *sender* and *receiver*, and *messages*, such as *plaintext* and *ciphertext*.

Example 5.3. '*knows*(*Alice*, $<m_1,\ m_2,\ \ldots,\ m_n>$)', '*sends*(*Alice*, *Bob*, $<m_1,\ m_2,\ \ldots,\ m_n>$)', '*generates*(*Alice*, $<m_1,\ m_2,\ \ldots,\ m_n>$)', and '*sees*(*Bob*, $<m_1,\ m_2,\ \ldots,\ m_n>$)' are atomic sentences. '*fresh*(m)' is regarded as an atomic sentence as well.

Atomic sentences can be logically combined with logical operators. And we are able to express facts that cannot be conveniently expressed by atomic sentences.

Suppose S_1 and S_2 are two atomic sentences. A *negation* is formed using the $\neg$ operator, such as $\neg S_1$ and $\neg S_2$. A *conjunction* is a set of sentences connected by the $\wedge$ operator, such as $S_1 \wedge S_2$. A *disjunction* is a set of sentences connected by the $\vee$ operator, such as $S_1 \vee S_2$. An *implication* is formed using the $\rightarrow$ operator, such as $S_1 \rightarrow S_2$. In particular, an *universally quantified sentence* is formed by combing the *universal quantifier* $\forall$, a variable ν, and any simpler sentence ψ. The intended meaning is that the sentence ψ is true, no matter what object the variable ν represents. Similarly, an *existentially quantified sentence* is formed by combing the *existential quantifier* $\exists$, a variable ν, and any simpler sentence ψ. The intended meaning is that the sentence ψ is true, for at least one object in the universe of disclosure.

There exists entailment relationship between the above operators. Furthermore, we have:

Generating:

$$\forall\ P, \vdash P\ \mathit{generates}\ m \rightarrow P\ \mathit{knows}\ m$$

states that if message m is generated by P, then P should know m.

Sending:

$$\forall\ P, \vdash P\ \mathit{knows}\ m \rightarrow P\ \mathit{sends}\ Q,\ m$$

states that if P knows message m, then P can send the message m to the receiver Q.

Receiving:

$$\forall\ P, \vdash P\ sends\ Q, m \rightarrow Q\ sees\ m$$

states that if P sends message m to Q, then Q can see message m.

According to the above relations, we can conclude $(\vdash generates(P, m) \rightarrow knows(P, m)) \wedge (\vdash knows(P, m) \rightarrow sends(P, Q, m)) \wedge (\vdash sends(P, Q, m) \rightarrow sees(Q, m)) \Rightarrow sees(Q, m)$. However, this cannot guarantee that the message m is reliable without regard to its freshness.

Rationality:

$$\forall\ Q, \vdash Q\ sees\ m \wedge fresh\ m \Rightarrow Q\ believes\ P, m$$

states that if Q sees message m from P and m is fresh, then it is reasonable to say Q believes m. P in fact indicates any principals who sent message m to Q.

From the above axioms, we can have the proposition below, which include the message generating, sending and receiving between principals to authenticate the message.

Proposition 5.1. *P generates message m and sends it to Q. If Q sees message m and m is fresh, then P believes Q in message m.*

$$\vdash P\ generates\ m \wedge P\ sends\ Q, m \wedge fresh\ m \Rightarrow Q\ believes\ P, m$$

where the principal P generates the message m and then sends it to the principal Q. If Q receives this message and confirms it is fresh, it is reasonable for principal Q to believe the message m sent from P. However, it does not imply principal Q believes the integrity and confidentiality of m, which need to be validated further by using other methods such as [29]. The *knows*, *sends* and *sees* operators actually represent the dynamics property of the generating, sending and receiving of secure messages. In addition, the message m can be a combination of messages, such as $m = \{\alpha, \beta\}$. Moreover, the implication $\alpha \rightarrow \beta$ is used to denote a special message called *inference rule*, in which the entailment relationship between α and β is defined.

In the following proof, the $\triangle$ on the right indicates that the associated sentences are in the initial database, and the number indicates the sentences from which the current sentence is derived.

Proof:

(1) $generates(P, m)$		$[\triangle]$
(2) $fresh(m)$		$[\triangle]$
(3)	$knows(P, m)$	[1, *Generating*]
(4)	$sends(P, Q, m)$	[3, *Sending*]
(5)	$sees(Q, m)$	[4, *Receiving*]
(6) $believes(Q, P, m)$		[2, 5, *Rationality*]

□

Definition 5.1. *Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of secure messages. The set of messages derived from sender, receiver and the third party are denoted by M_S, M_R and $M_T = \{M_{T_1}, \ldots, M_{T_k}\}$, respectively.*

The above sets of message can be viewed as principals' datasets. They are responsible for recording the messages that have been generated, sent or received between the principals. Ideally, it assumes that these messages should be consistent with each other in the aforementioned verification of security protocols.

Example 5.4. Suppose an electronic transaction may involve the *Buyer*, *Seller* and *Bank*. In general, the transaction has to be gradually authorized by *ATM*, *branch* and *head office*. The head office is viewed as the third party in this case. Therefore, we can obtain the set of exchanged messages using $M = \{M_{buyer}, M_{seller}, M_{ATM}, M_{branch}, M_{headoffice}\}$.

The users usually need to check its freshness after receiving a message. To achieve the objective, timestamps have been widely accepted by the formal analysis of security protocols.

Definition 5.2. *Let T be a timestamp attached to message m. If $|Clock - T| < \triangle t_1 + \triangle t_2$ regarding received messages or $T < T_{expiration}$ regarding generated messages then m is fresh; otherwise m is viewed as a replay.*

where the meaning of *Clock*, $\triangle t_1$ and $\triangle t_2$ can be seen in [40]. In addition, $T_{expiration}$ denotes the expiration time, which is designated to messages when they are generated. A timestamp serves three purposes: it can communicate the creator's local time, the creator's time zone, and the actual time in UTC (Universal Time Code). If the clock synchronization of both parties is difficult, a trusted third party can intervene as a notary and use its own clock as a reference [87, 143]. The timestamp plays an important role in preventing the replays of former transmitted secure messages.

Example 5.5. Suppose *Clock* is *8 Oct 2004 15:53:11 +0200*. The timestamp attached to a received message m is

8 Oct 2004 15:53:10 +0200

indicates that the creator's local time is *15:53:10*, that the creator's time zone is *+0200* (*two hours east of UTC*), and that the actual time is *8 Oct 2004 13:53:10* in UTC. Let $\triangle t_2 = 0.04$. Let server's clock be *8 Oct 2004 15:53:12 +0200*, then $\triangle t_1 = |12 - 11| = 1$. Thus, the message m is fresh due to $|Clock - T| = |10 - 11| = 1 < \triangle t_1 + \triangle t_2 = 1 + 0.04 = 1.04$.

Definition 5.3. *Let $\models_{support}$ be a* supporting *relationship. For a set of secure message M, $M \models_{support}$ is defined as follows, where α is an atom in $\mathcal{A}$, and each of them virtually denotes a message.*

$$\begin{cases} \mathrm{M}_S \models_{support} \alpha \textit{ iff } \text{`}\mathrm{M}_S \text{ knows } \alpha\text{''} \textit{ and `}\alpha \textit{ is fresh''} \\ \mathrm{M}_R \models_{support} \alpha \textit{ iff } \text{`}\mathrm{M}_R \text{ believes } \mathrm{M}_S, \alpha\text{''} \textit{ and `}\alpha \textit{ is fresh''} \\ \mathrm{M}_T \models_{support} \alpha \textit{ iff } \text{`}\mathrm{M}_T \text{ believes } \mathrm{M}_S, \alpha\text{''} \textit{ and `}\alpha \textit{ is fresh''} \end{cases}$$

where M_S, M_R and M_T represent the set of message of sender, receiver and the third party, respectively. The receiver and the third party can receive messages from different senders. As mentioned above, the sender decides whether a generated message α is fresh using $T < T_{expiration}$, whereas the receiver and the third party need to determine the freshness of α using $|Clock - T| < \triangle t_1 + \triangle t_2$ after receiving α from the sender.

Example 5.6. Suppose the timestamp attached to a generated message m by a sender S is

18 Sept 2004 10:38:18 +0200.

Let the expiration time $T_{expiration}$ be *18 Sept 2004 10:40:18 +0200.* Thus, m is fresh owing to $T < T_{expiration}$. Moreover, we have S generates $m \Rightarrow S$ knows m in terms of the **Generating axiom**. Therefore, we have S generates $m \wedge m$ is fresh $\Rightarrow S$ knows $m \wedge m$ is fresh $\Rightarrow S \models_{support} m$.

The supporting relationship takes into account not only the dynamics property but also the freshness property of secure messages. The former is realized by means of the *knows* and *sees* operators that provide user a useful way to describe the dynamic transmission of secure messages. And, the freshness of secure messages is protected by relying on the discriminant of timestamp.

As mentioned above, the rule presents an entailment relationship among messages. In particular, the conditions of a rule can be the conclusion of other rules.

Definition 5.4. *Suppose $\alpha_1, \ldots, \alpha_n$ $(n \geq 1)$ represent secure messages. Let $\alpha_1 \rightarrow \alpha_2$, $\alpha_2 \rightarrow \alpha_3$, $\ldots$, $\alpha_{n-1} \rightarrow \alpha_n$ be entailment relationships. Then we can deduce a new rule below if they are true.*

$$\alpha_1 \rightarrow \alpha_2 \wedge \alpha_2 \rightarrow \alpha_3 \wedge \ldots \wedge \alpha_{n-1} \rightarrow \alpha_n \Rightarrow \alpha_1 \rightarrow \alpha_n$$

These entailment relationships virtually denote the basic operation, such as encryption, decryption, signature and hashing of cryptography.

Example 5.7. 1) If *John* knows message m and *Alice*'s public key-exchange key $Kpb(Alice)$ then he knows $E(m, Kpb(Alice))$, which represents the message m was encrypted by $Kpb(Alice)$, namely '$knows(John, m) \wedge knows(John, Kpb(Alice)) \rightarrow knows(John, E(m, Kpb(Alice)))$'; and 2) if *John* knows message m encrypted by $Kpb(Alice)$, then he can send the encrypted m to *Alice*, namely '$knows(John, E(m, Kpb(Alice))) \rightarrow sends(John, Alice, E(m, Kpb(Alice)))$'. A new rule can then be derived from them. If *John* knows k and m, then he can send the encrypted m to *Alice* for the reason that the result of the first rule is actually the condition of the second rule. As a result, *John* is able to send the $E(m, k)$ to *Alice*, namely '$knows(John, m) \wedge knows(John, Kpb(Alice)) \rightarrow sends(John, Alice, E(m, Kpb(Alice)))$'.

In the same way, we have (1) '$knows(John,\ m \wedge k) \rightarrow sends(John,\ Alice,\ e(m,\ k))$'; (2) '$knows(John,\ m \wedge Spv(John)) \rightarrow sends(John,\ Alice,\ S(m,\ Spv(John)))$'; and (3) '$knows(John,\ m) \rightarrow sends(John,\ Alice,\ H(m))$', in which k and $Spv(John)$ indicate the symmetric key shared between *John* and *Alice* and *John*'s private signature key, respectively.

We define the relevant implication below in order to provide semantics basis for an analog to *modus ponens*. We do this by providing constraints. A *supporting relationship* should satisfy these constraints. Currently, there are just four constraints included. It is feasible for us to add more constraints if it is needed.

Let δ be a conjunction of atoms, and α and β be atoms in the usual sense. They have the following properties.

- *Conjunction constraint*: $M \models_{support} \alpha \wedge \beta$ iff $M \models_{support} \alpha$ and $M \models_{support} \beta$.
- *Implication constraint*: If $M \models_{support} \delta \rightarrow \alpha$ and $M \models_{support} \delta$, then $M \models_{support} \alpha$.
- *Transitivity constraint*: If $M \models_{support} \delta \rightarrow \alpha_i$ and $M \models_{support} \alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \beta$, then $M \models_{support} \alpha_1 \wedge \ldots \wedge \alpha_{i-1} \wedge \delta \wedge \alpha_{i+1} \wedge \ldots \wedge \alpha_n \rightarrow \beta$.
- *Negation constraint*: If $M \models_{support} \delta \rightarrow \neg\beta$, then $M \models_{support} \neg(\delta \rightarrow \beta)$.

5.2.3 Measuring Inconsistency in Secure Messages

In this section, we aim to provide a numerical estimation for the inconsistency in secure messages. We want to reflect each inconsistent set of formulae in a model, and then measure the inconsistency in the model. Obviously, this is not possible in classical logic, or indeed many non-classical logics, because there is no model of an inconsistent set of formulae [71]. Quasi-classical logic (QC logic) is motivated by the need to handle beliefs. It is intended to be a logic of beliefs in the "real world" rather than a logic of truths in the "real world". Models are based on a form of Herbrand interpretation.

Definition 5.5. *Let $\mathcal{A}$ be a set of atoms. Let $\mathcal{O}$ be a set of objects, where $+\alpha$ and $-\alpha$ represent positive object and negative object, respectively.*

$$\mathcal{O} = \{+\alpha \mid \alpha \in \mathcal{A}\} \cup \{-\alpha \mid \alpha \in \mathcal{A}\}$$

Example 5.8. Suppose '*The price is four thousand pounds*' is encrypted, in which a message block in cryptogram includes two letters, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}. Let $\alpha =$ '*four thousand pounds*'. The attacker can manipulate it so that only c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_{12}, c_{13}, c_{14} is received [130]. As a result, '*four thousand pounds*' and '*four pounds*' are viewed as a positive object and a negative object in a model, respectively.

This section gives a formal definition of measuring operator having a majority behaviour by merging individual beliefs together. For brevity, it assumes that the messages from message sources are proved to be fresh and reliable.

Definition 5.6. *Let $M \in \wp(\mathcal{L})$, $X \in \wp(\mathcal{A})$ and let $X \models_{support} M$ denote that $X \models_{support} \alpha$ holds for every α in M. The set of models of M is then defined as:*

$$model(M) = \{X \in \wp(\mathcal{A}) \mid X \models_{support} M\}$$

where M denotes a set of secure messages. The model of M actually represents a set of atoms that can support M.

As for each atom $\alpha \in M_R$ or $\alpha \in M_T$, $X \models_{support} \alpha$ means we can infer that the principal *believes* α and α *is fresh*. On the contrary, $X \not\models_{support} \alpha$ or $X \models_{support} \neg\alpha$ represents that the principal does not believe α. Given $\alpha \in M$, the conditions to support α can be seen in Section 5.2.2.

Example 5.9. Let $M = \{\alpha, \neg\beta, \gamma, \alpha \wedge \phi\}$, where $\alpha, \beta, \gamma, \phi \in \mathcal{A}$, and let $X = \{\alpha, -\beta, \alpha \wedge -\beta \rightarrow \gamma, \phi\}$. So $X \models_{support} \alpha$ and $X \models_{support} \neg\beta$. Also, $X \models_{support} \alpha \wedge \neg\beta \rightarrow \gamma$. Hence $X \models_{support} \gamma$, and $X \models_{support} \alpha \wedge \phi$. Hence every formula in M is supported by X, namely $X \models_{support} M$. Note that it assumes that all the elements supported by M are fresh, and are really generated or sent by the expected principals.

According to the above definition, it is observed that the *model* of a collection of messages is independent of the syntactic forms of the messages.

Proposition 5.2. *For all $i, j \geq 1$, $i \neq j$, $\exists\ M^i$ and $M^j \in \wp(\mathcal{L})$, if $M^i \equiv M^j$, then*

$$model(M^i) \equiv model(M^j)$$

Proof. Assume to be the contrary that $model(M^i) \not\equiv model(M^j)$. Then there exist $X \in \wp(\mathcal{A})$ and $X \models_{support} M^i$, such that $X \not\models_{support} M^j$. Let $\alpha \in X$ be an atom that can be inferred from M^i, namely $M^i \models_{support} \alpha$. Our task is to find a contradiction.

According to the above assumption, $X \not\models_{support} M^j$, it is possible that α can not be inferred from M^j, namely $M^j \not\models_{support} \alpha$. Hence it is natural to draw the conclusion $M^i \not\equiv M^j$.

This contradicts the fact that $M^i \equiv M^j$. □

This property is important for we are considering dealing with the inconsistency in secure messages with entailment relationship, and logically equivalent sets of message have the same messages.

Although the definition of models and supporting relationship provide us with basic concepts of inconsistency in secure messages, the set of models may include some irrelevant elements. In the last example, if $X' = X \cup \{\psi\}$, then $X' \models_{support} M$. Hence $X' \in model(M)$ in terms of the definition, in which the atom ψ is regarded as an irrelevant element to M for it does not belong to M. The irrelevant elements to M should not be considered in the model of M for it contributes nothing to measure the inconsistency in secure messages.

Let $Atoms(M)$ be the set of atom symbols in message source M, such as $Atoms(\{\alpha, -\beta, \gamma, \alpha \wedge \phi\}) = \{\alpha, -\beta, \gamma, \phi\}$. Suppose $\|M\|$ returns the

number of the set of atom symbols of M, such as $\|\{\alpha, -\beta, \gamma, \alpha \wedge \phi\}\| = 4$. In particular, if M is the union of several sets of messages, then the repeated atoms will be reserved and counted, such as $\|\{\alpha, \alpha, -\beta, \gamma, \alpha \wedge \phi\}\| = 5$.

In order to deal with inconsistency, we use equivalence models including no the irrelevant atoms.

Definition 5.7. *Let $M_P \in \wp(\mathcal{L})$ and let EQmodel(M_P) be the equivalence model for message source M_P, defined as follows:*

$$EQmodel(M_P) = \{X \in model(M_P) \mid \forall\, \alpha \in Atoms(M_P), \text{ then } \alpha \in X \text{ and } \|X\| = \|M_P\|\}$$

Example 5.10. Let α, β, γ and $\theta \in \mathcal{A}$ be atoms of message sources in the following sets of formulae .

(1) $EQmodel(\{\alpha, \beta, \alpha \rightarrow \gamma\}) \equiv \{+\alpha, +\beta, +\gamma\}$
(2) $EQmodel(\{\alpha \wedge \beta, -\gamma\}) \equiv \{+\alpha, +\beta, -\gamma\}$
(3) $EQmodel(\{\alpha, \alpha \rightarrow \beta, -\gamma, \beta \wedge -\gamma \rightarrow \theta\}) \equiv \{+\alpha, +\beta, -\gamma, +\theta\}$

In (1), $M = \{\alpha, \beta, \alpha \rightarrow \gamma\} \models_{support} \alpha$, $M \models_{support} \beta$, $M \models_{support} \alpha \rightarrow \gamma \Rightarrow M \models_{support} \gamma$ and $Atoms(\{\alpha, \beta, \alpha \rightarrow \gamma\}) = \{\alpha, \beta, \gamma\}$, so the equivalent model of M is $\{\alpha, \beta, \gamma\}$ in terms of the *implication constraint*; in (2), $M = \{\alpha \wedge \beta, -\gamma\} \models_{support} \alpha \wedge \beta \Rightarrow M \models_{support} \alpha$ and $M \models_{support} \beta$, $M \models_{support} -\gamma$, and $Atoms(\{\alpha \wedge \beta, -\gamma\}) = \{\alpha, \beta, -\gamma\}$, so we can infer the equivalent model $\{\alpha, \beta, -\gamma\}$; in (3), $M = \{\alpha, \alpha \rightarrow \beta, \neg\gamma, \beta \wedge \neg\gamma \rightarrow \theta\} \models_{support} \alpha$, $M \models_{support} \alpha \rightarrow \beta \Rightarrow M \models_{support} \beta$, $M \models_{support} \neg\gamma$, $M \models_{support} \beta \wedge \neg\gamma \rightarrow \theta \Rightarrow M \models_{support} \alpha \wedge \neg\gamma \Rightarrow M \models_{support} \theta$, and $Atoms(\{\alpha, \alpha \rightarrow \beta, \neg\gamma, \beta \wedge \neg\gamma \rightarrow \theta\}) = \{\alpha, \beta, -\gamma, \theta\}$, so we can conclude $\{\alpha, \beta, -\gamma, \theta\}$.

Proposition 5.3. *For all $i, j \geq 1$, $i \neq j$, $\exists\ M^i$ and $M^j \in \wp(\mathcal{L})$, if $M^i \equiv M^j$, then*

$$EQmodel(M^i) \equiv EQmodel(M^j)$$

Proof. Let $X \in model(M^i)$. Suppose $X_{ir}^- \in model(M^i)$ is the set of atoms, which does not include the irrelevant elements to $X \in \wp(\mathcal{A})$. Hence $X_{ir}^- \subseteq X$, and $\|X_{ir}^-\| = \|M^i\| = \|M^j\|$.

For M^i and $M^j \in \wp(\mathcal{L})$, if $M^i \equiv M^j$, then $model(M^i) \equiv model(M^j)$ in terms of the proved *Proposition 5.2*. Then, $\forall\ X \in model(M^i)$, $X \models_{support} M^i$ and $X \in model(M^j)$, $X \models_{support} M^j$.

Since $X_{ir}^- \subseteq X$, this means, for all X_{ir}^-, $X_{ir}^- \in model(M^i)$ and $X_{ir}^- \in model(M^j)$. Hence we can obtain $EQmodel(M^i) \equiv EQmodel(M^j)$ according to the *Definition 5.2*. □

The above description gives the definition with respect to the supporting relationship between secure messages and principals. To evaluate the degree of support for the secure messages, it is quantified by defining the cardinality of a supporting set of the messages.

Definition 5.8. *Let $\alpha \in \mathcal{A}$ be an atom. $|\alpha^M|$ is the total number of α supported by the equivalence model of M.*

Example 5.11. Let α and β be atoms. Let $M = \{\alpha, \alpha, \alpha \rightarrow \beta, \neg\gamma\}$. Hence we have $EQmodel(M) = \{+\alpha, +\alpha, +\beta, -\gamma\}$ according to the Definition 5.1 and Definition 5.2. Then we can obtain $|+\alpha^M| = 2$, $|+\beta^M| = 1$, and $|-\gamma^M| = 1$. Also, $|-\alpha^M| = |-\beta^M| = |+\gamma^M| = 0$ because $-\alpha$, $-\beta$ and $+\gamma$ are not found in the equivalence model of M.

Definition 5.9. *The support function from $\mathcal{A}$ to [0, 1] is defined below when α is not empty, and $|M \models_{support} \emptyset| = 0$.*

$$|M \models_{support} \alpha| = \frac{|+\alpha^M|}{|+\alpha^M| + |-\alpha^M|} \times 100$$

where $|+\alpha^M|$ is the number of occurrence of the set of α in the equivalence model of M. If $|M \models_{support} \alpha| = 0$, then we can say M has no opinion upon α and vice versa; if $|M \models_{support} \alpha| = 1$, this indicates that there is no negative object $-\alpha$ in the set of message M; if $|M \models_{support} \alpha| = c$, $0<c<1$, it represents that α is partially supported by M.

Example 5.12. Let α, β and γ be atoms. Suppose M is the same as Example 5.11. Since $EQmodel(M) = \{+\alpha, +\alpha, +\beta, -\gamma\}$, we can get $|M \models_{support} \alpha| = |+\alpha^M|/(|+\alpha^M| + |-\alpha^M|) = 2/(2 + 0) = 1$; $|M \models_{support} \neg\alpha| = 0$; $|M \models_{support} \beta| = 1/(1 + 0) = 1$; $|M \models_{support} \neg\beta| = 0$; $|M \models_{support} \gamma| = 0$; and $|M \models_{support} \neg\gamma| = 1/(1 + 0) = 1$.

In the last definition, we give a function to calculate the degree that a message source M supports α. Then the *reliability* between α and a set of message sources is defined below.

Definition 5.10. *The set of secure messages $\{M_S, M_R, M_T\}$ is defined as the sum of supports between α and each M_P, $P \in \{S, R, T\}$, and $|M_S \sqcup M_R \sqcup M_T \models_{support} \emptyset| = 0$.*

$$reliability(\alpha) = |M_S \sqcup M_R \sqcup M_T \models_{support} \alpha|$$

where the $\sqcup$ denotes a multiset union operation but the repeated items are reserved, such as $\{\alpha, \neg\beta\} \sqcup \{\alpha\} = \{\alpha, \alpha, \neg\beta\}$. Moreover α and β have to be fresh in each principal's message set in terms of the above definitions. In addition, if $|M_S \sqcup M_R \sqcup M_T \models_{support} \alpha| = 0$ then $reliability(\alpha) = 0$, which represents the principals do not support α.

In particular, if $reliability(\alpha) = reliability(\neg\alpha) = 50\%$, then M_S, M_R and M_T are fully conflicting with respect to α. In other words, the principals are *inconsistent* in regard to α.

Example 5.13. Let α, β, γ and θ be atoms. Let $M_S = \{\alpha, \beta, \gamma, \alpha \wedge \beta \wedge \gamma \rightarrow \theta\}$, $M_R = \{\alpha, \neg\beta, \gamma, \alpha \wedge \neg\beta \wedge \gamma \rightarrow \neg\theta\}$, and $M_T = \{\alpha, \beta, \neg\gamma, \neg\theta\}$. Then, $M_S \sqcup M_R \sqcup M_T = \{\alpha, \beta, \gamma, \alpha \wedge \beta \wedge \gamma \rightarrow \theta, \alpha, \neg\beta, \gamma, \alpha \wedge \neg\beta \wedge \gamma \rightarrow \neg\theta, \alpha, \beta, \neg\gamma, \neg\theta\}$. From the union of the message sources, we can infer $EQmodel(M_S \sqcup M_R \sqcup M_T) = \{+\alpha, +\alpha, +\alpha, +\beta, -\beta, +\beta, +\gamma, +\gamma, -\gamma, +\theta, -\theta, -\theta\}$. Hence $|+\alpha^{M_S \sqcup M_R \sqcup M_T}| = 3$, $|\beta^{M_S \sqcup M_R \sqcup M_T}| = 2$, $|-\beta^{M_S \sqcup M_R \sqcup M_T}| = 1$, $|+\gamma^{M_S \sqcup M_R \sqcup M_T}| = 2$, $|-\gamma^{M_S \sqcup M_R \sqcup M_T}| = 1$, $|+\theta^{M_S \sqcup M_R \sqcup M_T}| = 1$, and $|-\theta^{M_S \sqcup M_R \sqcup M_T}| = 2$. As a result, we have $reliability(\alpha) = 3/(3 + 0) = 1$, $reliability(\neg\alpha) = 0$, $reliability(\beta) = 2/(2 + 1)$ $2/3$, $reliability(\neg\beta) = 1/(2 + 1) = 1/3$, $reliability(\gamma) = 2/(2 + 1) = 2/3$, $reliability(\neg\gamma) = 1/(2 + 1) = 1/3$, $reliability(\theta) = 1/(2 + 1) = 1/3$ and $reliability(\neg\theta) = 2/(2 + 1) = 2/3$.

The above description assumes that the principals have equivalent degree of importance. However, they are usually associated with different weights in practical circumstances. For example, *CARoot*, *GCA*, and *CCA* in Figure 3.2. present a trust tree of certificate authorities, in which they have different levels of authority.

In this chapter, a special supporting relationship called weighted supporting is hence proposed where ϖ is a function that assigns each principal P a non-negative number representing the weight of P, $0 \leq \varpi \leq 1$. It is possible that S and R are occasionally treated with equivalent weight under the consideration of fairness. The weight function ϖ in fact represents the degree of importance of the principals. If $\varpi_A > \varpi_B$, $A \neq B$, it shows that A is more important than B, and more of its opinion will be reflected in the result of measuring the inconsistency in secure messages. In particular, if the weight of a principal is assigned zero, then it is deemed to be unreliable and its views will not be taken into account when computing the reliability.

Definition 5.11. *Let ϖ_S, ϖ_R and ϖ_T be the weight of M_S, M_R and M_T respectively, and let $|\alpha^i|$ be the sum of occurrence of the set of α in the message sources $i \in \{S, R, T\}$. The* weighted reliability *between α and the set of message sources $\{M_S, M_R, M_T\}$ is then defined as follows. In particular, if α is empty then $reliability(\emptyset, \varpi) = 0$.*

$$reliability(\alpha, \varpi) = \frac{\sum_{i \in \{S,R,T\}} |+\alpha^i| * \varpi_i}{\sum_{i \in \{S,R,T\}} (|+\alpha^i| + |-\alpha^i|) * \varpi_i}$$

The above method is adopted to quantify the support between an atom α (a message) and the set of message sources $\{M_S, M_R, M_T\}$. In this chapter, we define 50% as the minimum support threshold for reliability of any atoms, written as *minsupport*. In this scenario, the number of occurrence of α and $\neg\alpha$ are equivalent, so the sources of secure messages are definitely conflicting. We are just able to say the security of message α is uncertain even the reliability of α is over 0.5 but under 1.

Proposition 5.4. *Let M_i ($i \in [1, n]$) be a set of message sources and ϖ and $\varpi^{'}$ be two weight functions. If $\varpi_i^{'} = k * \varpi_i (k > 0)$ for $i = 1, \ldots, n$, then*

$$reliability(\alpha, \varpi) = reliability(\alpha, \varpi^{'})$$

Proof. Since $\varpi_i^{'} = k * \varpi_i$ $(k > 0)$, we have

$$\begin{aligned} reliability(\alpha, \varpi^{'}) &= \frac{\sum_{i\in[1,\ldots,n]} |+\alpha^i| * \varpi_i^{'}}{\sum_{i\in[1,\ldots,n]}(|+\alpha^i| + |-\alpha^i|) * \varpi_i^{'}} \\ &= \frac{\sum_{i\in[1,\ldots,n]} |+\alpha^i| * \varpi_i * k}{\sum_{i\in[1,\ldots,n]}(|+\alpha^i| + |-\alpha^i|) * \varpi_i * k} \\ &= \frac{\sum_{i\in[1,\ldots,n]} |+\alpha^i| * \varpi_i}{\sum_{i\in[1,\ldots,n]}(|+\alpha^i| + |-\alpha^i|) * \varpi_i} \end{aligned}$$

Hence, the proposition is true. □

Proposition 5.5. *Let α be an atom and let $M_i (i \in [1, n])$ be message sources and ϖ_i ($i \in [1, n]$) be its weight. Then, if the message source M_i are consistent in α and for all $i \in [1, n]$, $\varpi_i \neq 0$, then.*

$$reliability(\alpha, \varpi) = 1$$

Proof. Since the message sources M_i are consistent with respect to α, there is no negative object of α in the message sources. Hence we have $|-\alpha^i| = 0$ for all $i \in$ [1, n]. Then

$$\begin{aligned} reliability(\alpha, \varpi) &= \frac{\sum_{i\in[1,\ldots,n]} |+\alpha^i| * \varpi_i}{\sum_{i\in[1,\ldots,n]}(|+\alpha^i| + |-\alpha^i|) * \varpi_i} \\ &= \frac{\sum_{i\in[1,\ldots,n]} |+\alpha^i| * \varpi_i}{\sum_{i\in[1,\ldots,n]}(|+\alpha^i| + 0) * \varpi_i} \\ &= \frac{\sum_{i\in[1,\ldots,n]} |+\alpha^i| * \varpi_i}{\sum_{i\in[1,\ldots,n]}(|+\alpha^i|) * \varpi_i} = 1 \end{aligned}$$

Hence, the proposition is true. □

In general, if the reliability on message α is higher, it implies that the secure message sources have lower inconsistency in respect of the message α and vice versa.

The next consequence presents that the message sources that do not support α or are assigned the weight of *zero* can be discarded without influencing the result of computing the reliability on α.

Proposition 5.6. *Let $M_i (i \in [1, n])$ be message sources. Let α be an atom. If $M_i \not\models_{support} \alpha$ or $\varpi_i = 0$ then*

$$reliability(\alpha, \varpi) = \frac{\sum_{j \in [1,\ldots,i-1,i+1,\ldots,n]} |+\alpha^j| * \varpi_j}{\sum_{j \in [1,\ldots,i-1,i+1,\ldots,n]} (|+\alpha^j| + |-\alpha^j|) * \varpi_j}$$

Proof. If $\varpi_i = 0$, then obviously $|+\alpha^i| * \varpi_i = 0$ and $(|+\alpha^i| + |-\alpha^i|) * \varpi_i = 0$; if $M_i \not\models_{support} \alpha$, then the *reliability*$(\alpha)$ by M_i is equal to *zero* in terms of the definition of *reliability*. In either case, the opinion of M_i should be discarded from measuring the reliability. □

Based on the above definitions, the belief in secure messages is defined as follows.

Definition 5.12. *Let $\alpha \in \wp(\mathcal{A})$, and ϖ be the weight of message sources. The principal's belief of in α is defined as follows.*

$$belief(\alpha) = \begin{cases} secure & if \text{ reliability}(\alpha, \varpi) = 1 \\ insecure & if \text{ reliability}(\alpha, \varpi) \leq \text{minsupport} \\ uncertain & if \text{ reliability}(\alpha, \varpi) > \text{minsupport} \end{cases}$$

where the '*secure*' indicates that there is no negative object of α in the principals' messages, so we can say the principals are consistent with respect to α in the transaction; the '*insecure*' indicates the belief in α is completely inconsistent; and the '*uncertain*' indicates the message α is partially trusted, whereas the users have to make further verification to confirm its reliability.

Theorem 5.1. *Suppose $M_i (i \in S, S \subseteq [1, n])$ is consistent with respect to α and $\varpi_i \neq 0$. If $M_i \models_{support} \alpha$ and α is not mentioned by any other $M_j (j \in [1, n]$ and $j \neq i)$ then*

$$belief(\alpha) = \text{'secure'}$$

Proof. Since α is not included in M_j ($j \in$ [1, n] and $j \neq i$), M_j does not support α, namely $M_j \not\models_{support} \alpha$. Therefore, it is natural that the effect of M_j on the *reliability* of α can be ignored. According to *Proposition 5.6*, we have

$$reliability(\alpha, \varpi) = \frac{\sum_{i \in S} |+\alpha^i| * \varpi_i}{\sum_{i \in S} (|+\alpha^j| + |-\alpha^i|) * \varpi_i}$$

Since $M_i \models_{support} \alpha$, there is no negative object of α in them. Hence we have $|-\alpha^i| = 0$ for all $i \in S$, $S \subseteq$ [1, n]. Let M_i ($i \in S$) be a set of messages. Since $\varpi_i \neq 0$, we have

$$reliability(\alpha, \varpi) = 1$$

in terms of the *Proposition 5.5*. Hence we have $belief(\alpha) = secure$ according to *Definition 5.12*. From this the theorem follows. □

5.2.4 Examples of Measuring Inconsistency

For simplicity, it assumes that all the messages held by the secure message sources are fresh in the examples below. Also, the messages are assumed to be generated and sent by the sender and received and seen by whom it claims to be. Let α, β, γ, μ and θ be messages, which can be plaintext, ciphertext, symmetric key, signature key and so on.

Example 5.14. Suppose $M_S = \{\alpha, \beta, \alpha \wedge \beta \rightarrow \gamma\}$, $M_R = \{\neg\alpha, \beta, \neg\alpha \wedge \beta \rightarrow \neg\gamma\}$, and $M_T = \{\alpha, \beta, \gamma\}$. Let their weight be $\varpi(M_S) = \varpi(M_R) = 0.3$, but $\varpi(M_T) = 0.4$.

Then $EQmodel(M_S) \equiv EQmodel(M_T) \equiv \{+\alpha, +\beta, +\gamma\}$ and $EQmodel(M_R) \equiv \{-\alpha, +\beta, -\gamma\}$. Hence $M_S \sqcup M_R \sqcup M_T = \{\alpha, \beta, \gamma, \alpha, \beta, \gamma, -\alpha, \beta, -\gamma\}$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \alpha| = 2$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \neg\alpha| = 1$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \beta| = 3$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \gamma| = 2$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \neg\gamma| = 1$.

Finally, we can work out $reliability(\alpha, \varpi) = (0.3 + 0.4)/(0.3 + 0.3 + 0.4) = 0.7$, $reliability(\beta, \varpi) = (0.3 + 0.3 + 0.4)/(0.3 + 0.3 + 0.4) = 1$ and $reliability(\gamma, \varpi) = (0.3 + 0.4)/(0.3 + 0.3 + 0.4) = 0.7$. So $belief(\alpha) =$ '*uncertain*', $belief(\beta) =$ '*secure*' and $belief(\gamma) =$ '*uncertain*'.

In the set of messages M_S, $\alpha \wedge \beta$ is supported by M_S because M_S supports both α and β. The result indicates that β is secure since the belief in β is equal to 1. The reliability of α and γ needs to be further validated because the belief in them is over the minimal support 50% but under 1. To better understand this instance, α, β and γ can be viewed as message m, symmetric key k and encrypted message $e(m, k)$ respectively. Hence $\alpha \wedge \beta \rightarrow \gamma$ can be regarded as *P knows* $m \wedge$ *P knows* $k \rightarrow$ *P knows* $e(m, k)$.

Example 5.15. Suppose M_S, M_R and M_T are the same as the last example. Let $\varpi(M_S) = \varpi(M_R) = 0.2$, but $\varpi(M_T) = 0.6$.

Then, we have $reliability(\alpha, \varpi) = (0.2 + 0.6)/(0.2 + 0.2 + 0.6) = 0.8$, $reliability(\beta, \varpi) = 1$ and $reliability(\gamma, \varpi) = 0.8$ according to *Proposition 5.6*. Hence $belief(\alpha) =$ '*uncertain*', $belief(\beta) =$ '*secure*' and $belief(\gamma) =$ '*uncertain*' in terms of *Definition 5.12*.

In this case, the third party is assigned a higher weight than last example. Actually, it is usually reasonable to put more trust on the third party like trust centre. There is no change to the belief in β because the principals do not contain its negative object. On the other hand, the belief in α and γ increases because they receive more support form the principals.

Example 5.16. Suppose the weight of M_S, M_R and M_T keeps the same as Example 5.15. Let $M_S = \{\neg\alpha, \beta, \neg\alpha \wedge \beta \rightarrow \gamma\}$, $M_R = \{\neg\alpha, \neg\beta, \neg\alpha \wedge \neg\beta \rightarrow \neg\gamma\}$, and $M_T = \{\neg\alpha, \beta, \gamma\}$

Then we have $EQmodel(M_S) \equiv EQmodel(M_T) \equiv \{-\alpha, +\beta, +\gamma\}$ and $EQmodel(M_R) \equiv \{-\alpha, -\beta, -\gamma\}$ in terms of the constraints defined above. Hence $M_S \sqcup M_R \sqcup M_T = \{\neg\alpha, \beta, \gamma, \neg\alpha, \beta, \gamma, \neg\alpha, \neg\beta, \neg\gamma\}$, and $|M_S \sqcup M_R \sqcup M_T \models_{support} \alpha| = 0$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \neg\alpha| = 3$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \beta| = 2$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \neg\beta| = 1$, $|M_S \sqcup M_R \sqcup M_T \models_{support} \gamma| = 2$, and $|M_S \sqcup M_R \sqcup M_T \models_{support} \neg\gamma| = 1$.

In contrast to *Example 5.14*, we can see that no message source supports α. Hence it is reasonable to have $reliability(\alpha, \varpi) = 0$ according to *Proposition 5.6*. In addition, we can get $reliability(\beta, \varpi) = (0.2 + 0.6)/(0.2 + 0.6 + 0.2) = 0.8$, $reliability(\gamma, \varpi) = (0.2 + 0.6)/(0.2 + 0.6 + 0.2) = 0.8$. Therefore, we have $belief(\alpha) =$ '*insecure*', $belief(\beta) =$ '*uncertain*' and $belief(\gamma) =$ '*uncertain*' in terms of *Definition 5.12*.

Example 5.17. Let $M_S = \{\alpha \wedge \beta, \gamma\}$, $M_R = \{\neg\alpha \wedge \gamma\}$ and $M_T = \{\neg\alpha, \neg\beta, \gamma\}$. Let their weights be $\varpi(M_S) = 0.4$, $\varpi(M_R) = 0.2$ and $\varpi(M_T) = 0.4$.

Then $EQmodel(M_S) \equiv \{+\alpha, +\beta, +\gamma\}$, $EQmodel(M_R) \equiv \{-\alpha, +\gamma\}$ and $EQmodel(M_T) \equiv \{-\alpha, -\beta, +\gamma\}$. Hence $reliability(\alpha, \varpi) = 0.4/(0.4 + 0.2 + 0.4) = 0.4$, $reliability(\beta, \varpi) = 0.4/(0.4 + 0.4) = 0.5$ and $reliability(\gamma, \varpi) = 1$. As a result, we have $belief(\alpha) =$ '*insecure*', $belief(\beta) =$ '*insecure*' and $belief(\gamma) =$ '*secure*' according to *Proposition 5.6*.

The support for β from the receiver is regarded as *zero* because β is not included in M_R. The *zero* may indicate the receiver has not obtained this message. Hence the impact of M_R on β is discarded in terms of *Theorem 5.1*. However, the message β is still deemed to be insecure according to *Proposition 5.6*, though its reliability is exactly equal to 50%.

Example 5.18. Let $M_S = \{\alpha, \alpha \rightarrow \gamma, \beta, \theta, \beta \wedge \gamma \wedge \theta \rightarrow \mu\}$, $M_R = \{\neg\alpha, \neg\beta \wedge \neg\gamma, \theta, \neg\mu\}$, and $M_T = \{\neg\alpha, \beta \wedge \theta, \mu\}$. Suppose the weights of M_S, M_R and M_T remain the same with Example 5.17.

Then $EQmodel(M_S) \equiv \{+\alpha, +\beta, +\gamma, +\theta, +\mu\}$, $EQmodel(M_R) \equiv \{-\alpha, -\beta, -\gamma, +\theta, -\mu\}$ and $EQmodel(M_T) \equiv \{-\alpha, +\beta, +\theta, +\mu\}$. Hence we have $reliability(\alpha, \varpi) = 0.4/(0.4 + 0.2 + 0.4) = 0.4$, $reliability(\beta, \varpi) = (0.4 + 0.4)/(0.4 + 0.2 + 0.4) = 0.8$ and $reliability(\gamma, \varpi) = 0.4/(0.4 + 0.2) = 0.67$, $reliability(\theta, \varpi) = 1$ and $reliability(\mu, \varpi) = 0.8$. Therefore, $belief(\alpha) =$ '*insecure*', $belief(\beta) =$ '*uncertain*', $belief(\gamma) =$ '*uncertain*', $belief(\theta) =$ '*secure*' and $belief(\mu) =$ '*uncertain*'.

In this scenario, the message μ can be deduced by the *transitivity constraint* mentioned above, though it is an implicit message in M_S. Regardless of the high reliability 0.8 on both β and μ, the belief in β and μ is still uncertain. The high reliability of a message usually means that the inconsistency regarding this message is low. In particular, M_T does not support γ so its impact on the reliability of γ is discarded.

5.2.5 Experiments

We use some simulated transaction data to evaluate the inconsistency in secure messages. The used data corresponds to the cash withdrawal transaction from an Automated Teller Machine (ATM). When people make cash withdrawal from an ATM, they need to have knowledge of the related PIN. The customer places their card in the ATM slot and enters their PIN. Then the customer inputs the amount requested for withdrawal. The host computer needs to verify that the PIN is the proper one for that card. To ensure the amount dispensed at the machine is identical to the amount debited from the account, a sequence number is included on the response messages from host computer. Moreover, the encryption by DES algorithm protects the PIN being exposed to eavesdroppers who intercept the communications. It also protects PIN being read by the personnel who gain access to the bank's database.

Therefore, the transmitted messages in ATM transaction include *PIN* (encrypted PIN), *Key*(symmetric key), *Acct*(account number), *Amount* and *SN* (Sequence number). There are three principals including host computer, ATM, and the third party in this transaction, which are depicted as M_{host}, M_{ATM} and M_T, respectively.

- $M_{host} = \{PIN_{host},\ Key_{host},\ Acct_{host},\ Amount_{host},\ SN_{host},\ Weight_{host}\}$
- $M_{ATM} = \{PIN_{ATM}\ Key_{ATM}, Acct_{ATM}, Amount_{ATM}, SN_{ATM},\ Weight_{ATM}\}$
- $M_T = \{PIN_T,\ Key_T,\ Acct_T,\ Amount_T,\ SN_T,\ Weight_T\}$

where each item is assigned with values of *1*, *0* or *null* respectively. In particular, *null* value means that this item is empty in the set of messages. *1* and *0* represent two conflicting situations. For example, let $PIN_{host} = 1$, if encrypted PIN derived from ATM is not identical to the PIN in the host computer, then we said $PIN_{ATM} = 0$.

We have built an algorithm for measuring inconsistency in secure messages. It consists of three modules, *data_reading*, *data_transfer*, *reliability_output* and *belief_output*.

- the *data_reading* is used to collect the transaction data from the principals;
- the *data_transfer* assigns each secure message with *1*, *0* or *null* as described above;
- the *reliability_output* quantifies the reliability of each message and generates the *reliability* table; and
- the *belief_output* finally outputs the belief table in term of the generated *reliability* table.

Then the measuring algorithm is described below.

Input: the data regarding the cash withdraw transaction from ATM, including M_{host}, M_{ATM} and M_T.

Output: the reliability table and belief table.

class data_reading {
create Table M_{host} (*PIN*, *Key*, *Acct*, *Amount*, *SN*)
create Table M_{ATM} (*PIN*, *Key*, *Acct*, *Amount*, *SN*)
create Table M_T (*PIN*, *Key*, *Acct*, *Amount*, *SN*)
record = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*
from database of *Host*, *ATM* and *Third Party*)
insert into M_{host}, M_{ATM}, M_T *values* (record)
}

Table 5.1. Cash Withdraw from ATM

	PIN	*Key*	*Acct*	*Amount*	*SN*	*Weight*
host	1	1	1	1	1	0.4
ATM	1	0	0	0	null	0.3
T	1	null	0	1	1	0.3

class data_transfer {
field$_{host}$ = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*
from M_{host})
field$_{ATM}$ = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*
from M_{ATM})
field$_T$ = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*
from M_T)
Foreach field f in {*PIN*, *Key*, *Acct*, *Amount*, *SN*} do
{
if $f_{ATM} \neq$ null then
{
if $f_{ATM} = f_{host}$ then $f_{ATM} = 1$
else $f_{ATM} = 0$
}
if $f_T \neq$ null then
{
if $f_T = f_{host}$ then $f_T = 1$
else $f_T = 0$
}
if $f_{host} \neq$ null then $f_{host} = 1$
executeUpdate (*update* M_{host}, M_{ATM}, M_{host} set f_{host}, f_{ATM},
f_T)
}
}
class reliability_output {
executeUpdate (*alert table* M_{host}, M_{ATM}, M_T *ADD weight*)

Table 5.2. Reliability on ATM Transaction Data

	PIN	*Key*	*Acct*	*Amount*	*SN*
$reliability_{host \sqcup ATM \sqcup T}$	1	0.57	0.4	0.7	1

field$_{host}$ = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*, *weight* from M_{host})
field$_{ATM}$ = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*, *weight* from M_{ATM})
field$_T$ = *executeQuery* (*select PIN*, *Key*, *Acct*, *Amount*, *SN*, *weight* from M_T)
Foreach field f in {*PIN*, *Key*, *Acct*, *Amount*, *SN*} do
{
$S = \{host, ATM, T\}$
$reliability_f = 0$
Foreach $i \in \{host, ATM, T\}$ do
{
if f_i = null, then
$S = \{host, ATM, T\} - \{i\}$
}
Foreach $i \in S$ do
{
$reliability_f$ = *Sum* ($|f_i|$ * $weight_i$) / *Sum* (($|f_i|$ + | $- f_i|$) * $weight_i$) + $reliability_f$
}
executeUpdate (*insert into* M_{host}, M_{ATM}, M_{host} *values*($reliability_f$))
}
}
class belief_output {
Foreach field f in {*PIN*, *Key*, *Acct*, *Amount*, *SN*} do
{
if 0 $\leq reliability_f \leq$ 0.5 then
$belief_f$ = *'insecure'*
else
if 0.5 $< f_i <$ 1 then
$belief_f$ = *'uncertain'*
else
if f_i =1 then
$belief_f$ =*'secure'*
executeUpdate (*insert into* M_{host}, M_{ATM}, M_{host} *values*($belief_f$))
}
}

where the ATM, host computer and the third party should have the records of the transmitted data. To measure the inconsistency between the principals, it assumes that we are allowed to access the data. The obtained data is organized as the forms in Table 5.1.

Then, we can compute the *reliability* for each item, which is depicted in Table 5.2, in terms of the data derived from Table 5.1 and the function given in *Definition 5.11*.

Table 5.3. Belief in ATM Transaction Data

	PIN	*Key*	*Acct*	*Amount*	*SN*
$reliability_{host \sqcup ATM \sqcup T}$	1	0.57	0.4	0.7	1
$belief_{host \sqcup ATM \sqcup T}$	secure	uncertain	insecure	uncertain	secure

In Table 5.2, the reliability of encrypted PIN is 1, which indicates this item is reliable. Moreover, the reliability of null value (*empty*) is *zero* according to *Definition 5.11*. On the other hand, if the value of reliability on a message is bigger, then we can say the inconsistency of this message is lower and vice versa. This provides us an intuitive way to measure the inconsistency in secure messages.

Finally, the belief of secure messages is presented in Table 5.3 according to the given *reliability* in Table 5.2 and the discriminant function given in *Definition 5.12*.

According to the derived belief table, we can identify the uncertain messages from the secure and insecure messages, which are unreliable and need to be further validated. The results assist in guaranteeing the trust on the goal that can be put on the protocol.

5.3 Integration of Conflicting Beliefs in Secure Messages

The rapid growth of e-commerce (electronic commerce) on Internet brings out much important transaction information exchanged between principals. A number of messages are shared by more than one principal, whereas they may have inconsistent beliefs in these messages due to a hostile/uncertain environment. This discrepant belief may prevent us from representing the insecurity and uncertainty in a real trading situation. Most of existing formal analysis for security protocols however does not provide a way to handle the inconsistent beliefs in secure messages. This chapter thus proposes a numerical estimation to measure the inconsistent belief. It uses a probabilistic method that intuitively measures the belief from different principals based on a minimum trust that can be put on the goal of the protocol. We intend to merge the inconsistent beliefs by a weighted majority criteria. In addition, we present a

probabilistic semantics in combination with ENDL logic and apply the results to the protocol analysis.

5.3.1 Related Work

Recently, security protocols have played an important role in generating, delivering, encrypting and authenticating secure messages, and achieving security in distributed environment. A variety of formal methods and tools have been developed to analyse these protocols and uncover subtle flaws [22, 120]. The formal analysis assists in improving the performance of the protocol [22, 29, 62].

Usually, the above methods are classified into two groups in terms of their different purposes [24]: (1) is the group of approaches based on the belief of the principals [22], while (2) is the group of approaches based on the knowledge adversary. The former is used to evaluate the trust that can be put on the goal by the legitimate communicants using the beliefs of the principals [62, 131]. The latter analyses the security of a protocol by examining the knowledge gained by an intruder in the course of the protocol [117]. Although most of the impressive progress in protocol analysis has occurred in group (2), this still remained a fairly esoteric area until the publication of BAN logic [22]. That is why we will be focusing on the methods in (1).

The approaches in the first group regard the evolution of the beliefs of principals engaged in a protocol as a consequence of the protocol exchange. The verification usually starts from formalizing a set of exchanged messages, applying the inference rules, and then deducing the stated goal of the protocol. They aim at presenting how to start the initial set of beliefs, apply a set of inference rules, and finally achieve the main goals of the protocols can be deduced. They provide short, clear and formal proofs of these goals and detect flaws in security protocols. ENDL logic [29] is in this group and is especially designed for the verification of electronic transaction protocols.

In the previous protocol analysis, the communication channel and principals are ideally assumed to be secure and trustworthy. However, in a hostile and/or uncertain environment, the beliefs of principals and transmitted messages can no longer be justified. For example, a customer wants to order a gift valued at $20 but the merchant may receive a tampered order valued at $50 from an intruder. In other words, the principals may have inconsistent beliefs in secure messages. Then, it is necessary to have the capability of modelling the imperfect working conditions and verifying the protocol under such circumstances. Recently, the uncertainty and partial belief have attracted much attentions in developing methods [24, 106]. Many different formalisms and methodologies have been proposed for handling uncertain knowledge. Most of them are directly or indirectly based on probability theory. However, none of them have touched upon the key issue of measuring the inconsistent beliefs in secure messages.

The study of the connections between probability and logic has a long history, such as a new probabilistic method to deal with uncertainty and partial belief [51]. In [66], they discussed two useful and quite different ways of understanding belief functions that consist of a generalized probability function and a way of representing evidence. Campbell, Safavi-Naini, and Pleasants [24] presented an extension of BAN logic to reason about a secure protocol by attaching the probabilities to the corresponding statements in the logic and to calculate a measure of trust in the goal of a protocol, which quantifies the beliefs of principals. It puts emphasis upon the beliefs of principals. However, the required inexact reasoning is computationally expensive. This method can have difficulty to analyse finance protocols due to its lack of semantics, such as freshness. They do not discuss how to integrate the inconsistent beliefs between principals. Thus, it requires us to extend or adapt current methods to deal with the inconsistent belief, offer a numeric estimation to the belief and enhance the formal proof of the protocol.

In recent years, many formalisms have been put forward in the literatures to cope with the inconsistent messages of knowledge bases. Liberatore and Schaerf gave a merging process arbitration to merge the different views between different sources of information and investigates the properties any arbitration operator should satisfy [93]. It intends to preserve as much information as possible after merging; Lin and Mendelzon defined a merging operator by majority in contrast with arbitration to merge knowledge base in [96], and gave formal semantics for merging multiple knowledge bases with weights by using the method of Dalal [39] in [95]; Konieczny and Pino Perez considered the problem of merging several belief bases in the presence of integrity constrains [89], and explored the frontier between merging operator and arbitration operator in [90]; Hunter presented a framework for evaluating the significance of inconsistencies in [72]. The above work is a good foundation to handle conflicting secure messages.

From the above observations, (1) most of the current work focuses on the uncertainty of belief rather than measuring the inconsistent beliefs in secure messages; (2) the traditional methods to deal with uncertain belief using probability however have not taken into account the properties of secure messages.

In an open and distributed environment, the participants may have inconsistent, even conflicting belief in secure messages. In addition, the beliefs are not permanent but just valid within the period of validity. The traditional logics for protocol analysis cannot deal with the belief in secure messages and intuitively measure the trust in the goal of the protocol. Therefore, it is necessary to develop a novel method to measure the partially conflicting belief in secure messages to ensure correct protocol analysis.

This section aims in handling partially conflicting beliefs in secure messages and enhancing the formal proof of security protocols.

5.3.2 Basic Concepts

In this section, we describe some technical preliminaries. We first present the relationship between probability and formal proof. And then we present the semantics definition of the formal framework.

Formal logic, probability and belief. Some recent work of probabilistic logic can be found in [24, 51]. We will be concerned here with the probability assigned to a set of assumptions and inference rules and relate them to the probability of conclusions that can be derived from them. These probabilities are useful to express less than certain assumptions and rules in the formal proof of the correctness of protocols. Furthermore, this helps deal with partially conflicting belief in secure messages and correctly measure the trust that can be put on the goal of the protocol.

A formal logic used to verify security protocols includes a collection of sentences and inference rules. For example, '*knows* (*Alice*, k)' and '*sends*(*Alice*, *Bob*, m)' are sentences. The sentences are formed in terms of the syntax of the logic. It generates meaningful statements according to its semantics. An inference rule actually indicates the relationship between a collection of sentences.

Let $\mathcal{S}$ be the set of sentences in the formal logic and $\mathcal{S}^+$ be the closure of $\mathcal{S}$ which represents the set of whole sentences derived from the $\mathcal{S}$ by applying the rules of the logic. A set of rules $\mathcal{R}$ can be defined as the form of $a_1, a_2, \ldots, a_n \Rightarrow c$, which includes the sentence $c \in \mathcal{S}^+$ in conjunction with the sentences $a_1, a_2, \ldots, a_n$. Let $s_1, s_2, \ldots, s_k$ be a proof of a sentence c, in which each s_i ($1 \leq s_i \leq$ k) can be an axiom or is inferred from available sentences in sequence by a rule instance in $\mathcal{R}$.

A sentence ϕ can be true or false. We can imagine two sets of *possible worlds* $\mathcal{W}_1$ and $\mathcal{W}_2$. $\mathcal{W}_1$ contains *worlds* in which ϕ is true and $\mathcal{W}_2$ contains *worlds* where ϕ is false. The actual world, however, must be in one of these two sets, but we might not know which one. As a result, we need to model this less than certain situations using probability.

Let $\mathcal{B} = \mathcal{S} \cup \mathcal{R}$. Define the probability space ($\mathcal{W}$, $\mathcal{F}$, $\mathcal{P}$) for $\mathcal{B}$, in which the sample space $\mathcal{W}$ is a nonempty set and is known as outcomes, the events $\mathcal{F}$ denote sets of outcomes, and the probability measure $\mathcal{P}$ is a function that assigns to each event a probability between 0 and 1. Consequently, for each $w \in \mathcal{W}$, there is a truth assignment $a \mapsto w(a) = 0$ or 1 defined on $\mathcal{B}$, the probability of $a \in \mathcal{B}$ by

$$p(a) = p\ (\{w : w(a) = 1\}).$$

Let c be any conclusions that can be inferred from $\mathcal{B}$. We can extend the truth assignment to c by defining $w(c) = 1$ only if there is a proof of c from $\mathcal{B}$. In other words, we have $w(a) = 1$ for each sentence or inference rule a used in this proof. The probability of a conclusion $c \in \mathcal{A}$ can then be defined by

$$p(c) = p\ (\{w : w(c) = 1\}).$$

The set $\mathcal{W}$ actually represents the state of the world. In each state, some of the sentences in $\mathcal{S}$ and inference rules are valid. The probability P describes the degree of belief in the conclusions. The probability of a conclusion c can be defined as the probability that the proof of c from $\mathcal{B}$ is valid. It actually depends on the probabilities that the assumptions and rules are trustworthy. Ideally, they were assumed to be certain in the previous formal proof of protocols. However, the hostile/uncertain environments can result in inconsistent beliefs in secure messages. Therefore, it is necessary to attach probabilities to rules to model such insecure environments.

Let P be a principal, let M_P be a set of secure messages known by P and let $M = \{M_{P_1}, M_{P_2}, \ldots, M_{P_n}\}$. It is possible that two principals P_i and P_j can share some messages with each other. Usually, a weight is assigned to each principal in terms of its priorities. If a principal has a higher weight, more his/her opinion will be reflected in measuring the belief.

It is observed that the belief in a goal c depends on the *assumed belief* (*weight*), *observed belief* and the *probability* of the rule. In particular, there might be more than one rule that can be used to verify a goal of the protocol. In that case, we may have to examine whether the rules have an intersection of conditions. If so, more complicated probability computation is required, otherwise they will be viewed as independent rules. Note that the sender does not need to use inference rules to authenticate c. The details of measuring the belief can be seen in the next section.

The derived inconsistent beliefs prevent the user from deciding the trust in the goal of the protocol correctly. Therefore, it is critical to merge the beliefs, and ensure the correctness of the formal proof of the protocol.

Semantic definition. A principal is a main participant in a protocol. A set of secure messages M is a finite set of sentences. We say M supports a sentence α if M implies α and trusts it (i.e., $M \models_{support} \alpha$), and M opposes α if M implies $\neg\alpha$ ($M \models_{support} \neg\alpha$) and distrusts it.

Suppose $P_1, P_2, \cdots, P_n$ ($n \geq 1$) are the principals to be involved in the protocol and ϖ is a function that assigns each of the principals a non-negative number representing the weight of the principal. The weight function ϖ is intended to capture the relative degree of importance of the principals. If $\varpi(P_i)$ is higher, P_i is more important within the group of principals, and we want more of its knowledge to be reflected in the result of trust in the goal of the protocol. The smallest number that can be assigned to a principal is zero. Intuitively, if a principal is assigned a zero weight then its opinion is discarded while measuring the belief.

A message may be encrypted in a number of ways, such as encrypted by a symmetric key; a private/public key. In particular, the encryption may include additional components such as the hashing of a message m and a timestamp. Moreover, we have the following constructs defined in [29]:

- $<m_1, \cdots, m_n>$ represents the combination of a set of messages, including $m_1, \cdots, m_n$.
- **sends**(X, Y, m) represents the messages m was sent from principal X to principal Y.
- **knows**(X, m) represents principal X knows the message m.
- **sees**(X, m) represents principal X receives the message m.
- **fresh**(m) represents the message m is fresh.
- **believes**(X, Y, m) represents principal X believes the message m is fresh and really from Y.
- **authenticates**(X, Y, m) represents principal X believes that the message from principal Y is authentic.
- $\{m\}_K$ represents the message m is encrypted using the key K.
- K_{XY} represents the key K is shared between principal X and principal Y.

Example 5.19. Suppose m, m_1 and m_2 are messages and X_1 and X_2 are principals. $<m_1, m_2>$ denotes the concatenation of m_1 and m_2; "**sends**(X_1, X_2, m)" denotes the message m was sent from X_1 to X_2; "**knows**(X_1, m)" denotes m is known by X_1; "**sees**(X_2, m)" represents principal X_2 receives message m; "**fresh**(m)" represents m is fresh and not a replay of previous messages; "**believes**(X_1, X_2, m)" represents X_1 believes that the message m is fresh and really from X_2; "**authenticates**(X_1, X_2, m)" represents X_1 believes that the message m from X_2 is reliable; $K_{X_1X_2}$ represents the key K is shared by X_1 and X_2.

As the symbols describe, the checking of a message authenticity could be viewed as the combination of a series of constructs, such as '*believes*', '*fresh*', '*see*'. In [22], they make a realistic assumption that each principal can recognize and ignore his own messages; the originator of each message is included for this purpose. However, this cannot exclude the possibility that different principals may have inconsistent beliefs in the message. Unfortunately, this less than certain assumptions and inference rules below cannot be represented by the operational semantics of ENDL logic. Thus, its semantics need to be extended. The details can be seen below.

In general, facts are stated in the form of expressions called *sentences*, or sometimes well-formed formulae. We define that an *atomic sentence* is formed from a n-ary relation operator mentioned above and n atom symbols a_1, a_2, ..., a_n, by combining them as follows.

$$r\ (a_1,\ a_2,\ \ldots,\ a_n)$$

We do not consider the atomic sentences involving mathematical relations due to the freshness and dynamics properties of secure messages.

Example 5.20. "**knows**(*Alice*, $<m_1, m_2, \ldots, m_n>$)" and "**sends**(*Alice*, *Bob*, $<m_1, m_2, \ldots, m_n>$)" are both atomic sentences. "**fresh**(m)" is viewed as an atomic sentence as well.

We also want to express facts that cannot be conveniently expressed by atomic sentences. The atomic sentences can logically be combined by logical operators.

Suppose S_1 and S_2 are two atomic sentences. A *negation* is formed using the $\neg$ operator, such as $\neg S_1$ and $\neg S_2$. A *conjunction* is a set of sentences connected by the $\wedge$ operator, such as $S_1 \wedge S_2$. A *disjunction* sentences is a set of sentences connected by the $\vee$ operator, such as $S_1 \vee S_2$. An *implication* is formed using the $\rightarrow$ operator, such as $S_1 \rightarrow S_2$. In particular, an *existentially quantified sentence* is formed by combing the *existential quantifier* $\exists$, a variable ν, and any simpler sentence ψ. The intended meaning is that the sentence ψ is true, for at least one object in the universe of disclosure.

The semantics of ENDL can be extended to allow for uncertain assumptions and inference rules in the proof of the correctness of the protocol. According to existing operational semantics of ENDL, principals have some initial beliefs when starting the protocol and trust which evolve in the course of the protocol in terms of certain rules. These rules are abstraction of inference processes and assist in determining the final beliefs of the principals.

Sentences in the ENDL can generate various statements. The first type of sentences is the atomic sentence described above such as the *sends*() and *sees*(). It shows what knowledge are available to the principals within a transaction. In [24], these sentences are assigned a generalised truth value 1 in all possible states. However, the messages may not be always consistent in all states but can be missing or tampered in a hostile environment. For example, a customer may send a \$10 order for a *flash disk*, but the merchant sees \$20 order instead. It is unreasonable to continuously assign a truth value 1 to the sentences since the principals have inconsistent beliefs in them. In this chapter, we use the support of sentences as the truth value. The details can be seen in Section 5.3.3

The second type of sentences include the axioms of ENDL, which represent the basic entailment relation with respect to encryption, decryption, key allocation, signature and authentication. For example, the principal P should know its own private key. The sentences below present the fundamental entailment relations of ENDL.

(1) $\vdash knows(P, m) \rightarrow sends(P, Q, m)$.

It means that if the principal P knows m, then P can sends m to another principal Q.

(2) $\vdash sends(P, Q, m) \rightarrow sees(Q, m)$

It means that if P sends m to Q, then Q will see m.

As a consequence, we can conclude $knows(P, m) \wedge (\vdash knows(P, m) \rightarrow sends(P, Q, m)) \wedge (\vdash sends(P, Q, m) \rightarrow sees(Q, m)) \Rightarrow sees(Q, m)$.

We will assign the generalised truth value 1 to similar sentences since they have been proved to be true.

In ENDL logic [29], *sends*, *knows*, *sees* and *fresh* are primitive operators. They can be represented as a compact form, namely rules.

$\vdash knows(P,\ m) \times sends(P,\ Q,\ m) \times fresh(m) \times sees(Q,\ m) \Rightarrow believes(Q,\ P,\ m)$

where the principal P generates the message m and then sends it to the principal Q. If Q receives this message and confirms that it is fresh, it is reasonable for principal Q to believe the message m from P. However, it does not imply that principal Q believes the integrity and confidentiality of m. They need to be further verified according to the inference rules below.

There are three main rules in ENDL that are involved in the authentication of messages. They are derived from the authentication axioms of ENDL and have been proved to be true.

(1) $\vdash knows(X,\ m) \times knows(X,\ S(<ID_Y,\ T,\ H(m)>,\ Spv(Y))) \times knows(X,\ Spb(Y)) \Rightarrow authenticates(X,\ Y,\ m)$.
(2) $\vdash knows(X,\ m) \times authenticates(X,\ Y,\ H(m)) \Rightarrow authenticates(X,\ Y,\ m)$.
(3) $\vdash knows(X,\ Spb(Y)) \times knows(X,\ S(<ID_Y,\ T,\ Spb(Y)>,\ Spv(CA))) \times knows(X,\ Spb(CA)) \Rightarrow authenticates(X,\ Y,\ Spb(Y))$.

where ID_Y is Y's identity; T is the timestamp; $H(m)$ is the hashing of message m; and $Spv(Y)$ and $Spb(Y)$ represent Y's private and public signature key, respectively. The definition and use of timestamp can be found in [40].

Inference rules of ENDL play an important role in generating new beliefs according to the existing beliefs of principals and their available messages during the transaction. The probability that a rule holds can be defined as the probability that given that the beliefs of the conditions of the rule are true, then the belief of the conclusion of the rule is also true. Thus, it is viewed as a conditional probability of the observed belief of the principal.

The aforementioned inconsistent beliefs increase the complexity because we have to consider all potential factors that may have an effect on measuring the beliefs. Moreover, there are many situations in which humans possess and reason with uncertain belief and allow for degrees of certainty in the truth of a proposition. To extend the apparatus of first-order logic to permit the use of probability theory in reasoning about uncertain statements, we must connect the idea of a *sentence* in logic with the idea of a *random variable* in probability theory.

Let $M_P = \{m_1, m_2, \ldots, m_n\}$ be a collection of secure message of principal P. Each message may be shared by more than one principal simultaneously since it might be transmitted between principals in a trading process. Usually, we classify the messages into three categories M_S, M_R and M_T, respectively,. These actually represent the messages recorded in corresponding principal's computer system. Ideally, it assumes that the principals should have equal weight to a given message.

Definition 5.13. *Let* $\models_{support}$ *be a supporting relationship. For a set of secure message M,* $M \models_{support}$ *is defined as follows, where* α *is an atom in* $\mathcal{A}$*, and each of them virtually denotes a message.*

$$\begin{cases} \mathrm{M}_S \models_{support} \alpha \textit{ iff } \text{`knows}(\mathrm{S}, \alpha)\text{'} \wedge \text{`fresh}(\alpha)\text{'} \\ \mathrm{M}_R \models_{support} \alpha \textit{ iff } \text{`believes}(\mathrm{R}, \mathrm{S}, \alpha)\text{'} \wedge \text{`fresh}(\alpha)\text{'} \\ \mathrm{M}_T \models_{support} \alpha \textit{ iff } \text{`believes}(\mathrm{T}, \mathrm{S}, \alpha)\text{'} \wedge \text{`fresh}(\alpha)\text{'} \end{cases}$$

where M_S, M_R and M_T denote the set of messages of sender, receiver and the third party, respectively. In particular, "α *is fresh*" in these formulae is decided by using $T < T_{expiration}$. On the other hand, R and T need to check the freshness of α by using $|Clock - T| < \triangle t_1 + \triangle t_2$ when they receive α from the sender [40]. These will guarantee the message α is not a replay and is really from the the expected principal.

Definition 5.14. *Suppose* $\beta_1, \ldots, \beta_n$ $(\mathrm{n} \geq 1)$ *are secure messages.* $M \in \{M_S, M_R, M_T\}$*. Let* $\beta_1 \wedge \ldots \wedge \beta_n$ *be a conjunction, which is a set of secure messages connected by the* $\wedge$ *operator.*

$$M \models_{support} \beta_1 \wedge \ldots \wedge M \models_{support} \beta_n \text{ iff } M \models_{support} \beta_1 \wedge \ldots \wedge \beta_n$$

This means that, if M *supports* every message β_i, $1 \leq i \leq n$, then M should support the conjunction of these messages connected by the $\wedge$ operator and vice versa.

The supporting relation considers the dynamics property of secure message by using the *knows* and *sees* operators. Furthermore, the freshness of secure message is ensured by the timestamp.

Definition 5.15. *Let* $\models_{match}$ *be a matching relationship. For a set of secure messages M,* $M \models_{match}$ *is defined as follows, where R is a set of rules used to authenticate messages.*

$$\begin{cases} \mathrm{M}_R \models_{match} \mathrm{r} \textit{ iff } \exists\ \mathrm{M}_R{}' \subseteq \mathrm{M}_R, \exists\ \mathrm{r} \in \mathrm{R}\text{' and '} \mathrm{M}_R{}' \text{ matches r}\text{'} \\ \mathrm{M}_T \models_{match} \mathrm{r} \textit{ iff } \exists\ \mathrm{M}_T{}' \subseteq \mathrm{M}_T, \exists\ \mathrm{r} \in \mathrm{R}\text{' and '} \mathrm{M}_T{}' \text{ matches r}\text{'} \end{cases}$$

where the principal intends to find messages required to match the rule. If there is an inference rule that can be satisfied by the obtained messages, it is true; otherwise false. S is the initiator of the message. Therefore he/she just verifies the message m in terms of the supporting relation defined above but does not need to match the inference rules.

Definition 5.16. *Let* $\models_{auth}$ *be a belief relationship. For a set of secure message M,* $M \models_{auth}$ *m is defined as follows, where m is a message needed to be authenticated.*

$$\begin{cases} \mathrm{M}_S \models_{auth} \mathrm{m} \textit{ iff } \text{`}\mathrm{M}_S \models_{support} \mathrm{m}\text{'} \\ \mathrm{M}_R \models_{auth} \mathrm{m} \textit{ iff } \text{`}\mathrm{M}_R \models_{support} \mathrm{m}\text{' and '} \exists \mathrm{r} \in \mathrm{R}, \mathrm{M}_R \models_{match} \mathrm{r}\text{'} \\ \mathrm{M}_T \models_{auth} \mathrm{m} \textit{ iff } \text{`}\mathrm{M}_T \models_{support} \mathrm{m}\text{' and '} \exists \mathrm{r} \in \mathrm{R}, \mathrm{M}_T \models_{match} \mathrm{r}\text{'} \end{cases}$$

where the supporting relation is the starting point, from which the principals can go ahead to match the rules mentioned above. If a rule is satisfied we can say the authenticated message m is reliable; otherwise unreliable.

The $\models_{support}$, $\models_{match}$ and $\models_{auth}$ actually represent the transformation of beliefs during the authentication process of messages. They all assist in authenticating the message and will be operated in order during the authentication.

Similarly, the above supporting relationships can satisfy some constraints. The details can be seen in Section 5.2.2

5.3.3 Handling Inconsistent Beliefs in Secure Messages

Belief in secure messages. Usually, multi-Agent systems or heterogeneous database are natural environment for the occurrence of inconsistent beliefs. In particular, principals often have contradictory beliefs in secure messages in a hostile/uncertain environment. We are interested in the general case when their individual belief assignment to messages is inconsistent; and we are interested in finding an intuitive way to measure the trust that can be put on the goal of the protocol.

Belief is referred as a principal's view of secure messages, which can be introduced directly or inferred through perception, assumption or communication between principals rather than an individual.

A belief *bel* in secure message m from principal P is represented by a tuple

$$<m,\ bel_P,\ \mathcal{E}(bel_P),\ \mathcal{F}(bel_P),\ P>$$

where m is a secure message, bel_P identifies the belief, $\mathcal{E}(bel_P)$ specifies how the inferred belief is derived (assumed, observed, and probability of the belief of inference rules), $\mathcal{F}(bel_P)$ is the set of foundations that support the belief, and P identifies the principal of the belief, including S, R and T. The belief status is established in terms of the $\mathcal{F}(bel_P)$. If $\mathcal{F}(bel_P) = \emptyset$, then m is disbelieved; otherwise, if $\mathcal{F}(bel_P) \neq \emptyset$, the belief status of m is uncertain.

The assumed belief is a form of subjective assignment of probability. However, it is necessary to take into account the objective assignment, including the observed belief and the belief transmission when using inference rules, to evaluate the performance of the protocol correctly. In the following sections, we attempt to measure the beliefs and merge them.

Quality of support using minimal QC model. In this section, we will define and discuss QC minimal model of a collection of messages of principals, and measure the observed belief of principals.

QC [71] logic is motivated by the need to deal with belief. The definition of QC model can be seen in Section 5.2.3.

To measure the inconsistency, the minimal QC model (MQC) is often used. A minimal QC model of M (a collection of secure messages) is a QC model X of M such that for every subset $Y \subset X$, $Y \notin \mathrm{QC}(X)$.

We can use an example to illustrate how to generate minimal QC model.

Example 5.21. Suppose $M_1 = \{\alpha, \beta \vee \gamma\}$ and $M_2 = \{\alpha \vee \beta, \neg\alpha \vee \gamma\}$. We have

$\mathrm{MQC}(M_1) = \{\{+\alpha, +\beta\}, \{+\alpha, +\gamma\}\}$
$\mathrm{MQC}(M_2) = \{\{+\alpha, +\gamma\}, \{-\alpha, +\beta\}, \{+\beta, +\gamma\}\}$

In this example, M_1 has two MQCs, and M_2 has three MQCs. Thus, we may obtain multiple results when calculating the support of principals for a sentence by using different MQCs. In that case, their mean is calculated.

Definition 5.17. *Let M be a set of secure messages and let X be a MQC of M. The support function from $\mathcal{A}$ to $[0, 1]$ is defined below when α is not empty, and $supp(M, \emptyset) = 0$.*

$$supp(X, \alpha) = \frac{|+\alpha|}{|+\alpha \cup -\alpha|} \times 100 \tag{5.1}$$

where $supp(X, \alpha)$ is the total number of α supported by the minimal QC model X of M and $|\alpha|$ is the number of occurrence of the set of α in X. If $supp(X, \alpha) = 0$, then we can say X has no opinion upon α and vice versa; if $supp(X, \alpha) = 1$, it indicates that there is no negative object $-\alpha$ in X; if $supp(X, \alpha) = c$, $0<c<1$, it represents α is partially supported by X.

The above definition actually provides a function to calculate the degree that a MQC model supports α. The support between α and M is defined below.

Definition 5.18. *Suppose $X_1, X_2, \cdots, X_k$ are all MQC models of M. The support between M and a sentence α is defined as the mean of all $supp(X_i, \alpha)$.*

$$supp(M, \alpha) = \sum_{i=1}^{k} supp(X_i, \alpha)/k \tag{5.2}$$

where some models may not support α. In particular, if $supp(M, \alpha) = 0$, it indicates that α is not supported by M at all.

Note, we must check whether α is fresh or not like Definition 5.2. If α may be identified as a replay, it will be reported to the user, rather than calculating the support.

The obtained support represents the observed belief. To decide the trust in the goal, we need to consider the weight of principals and the probability of inference rules together.

Measuring inconsistent beliefs. A numeric estimation is defined below to measure the inconsistent beliefs in secure messages.

In this chapter, we aim to derive the reliability of statement "X authenticates Y, m", namely that in X's view the message m sent from Y is authentic,

in which the message m can be a plain text, cipher text or encryption key, and so on. The satisfiable conditions of secure messages can be seen in Definition 5.3.

Definition 5.19. *Let m be the authenticated goal, let $Bel_P(m)$ be the belief of the principal P in m, $P \in \{S.\ R,\ T\}$ and let $p_R(r)$ be the probability of rules that can be used by P to authenticate m. The belief of principals in a statement m can be defined as follows.*

$$\begin{cases} \mathrm{Bel}_S(\mathrm{m}) = \mathrm{supp}(\mathrm{M}_S,\ \mathrm{m}) * \varpi_S \\ \mathrm{Bel}_R(\mathrm{m}) = \mathrm{p}_R(\mathrm{r}) * \varpi_R \\ \mathrm{Bel}_T(\mathrm{m}) = \mathrm{p}_T(\mathrm{r}) * \varpi_T \end{cases}$$

where ϖ_S, ϖ_R and ϖ_T represent the assumed belief of the sender, receiver and the third party, respectively; as mentioned above, the sender does not need to use inference rule to authenticate m, so only the assumed belief and observed belief ($supp(M_S, m)$) are considered here; the probabilities of $p_R(r)$ and $p_T(r)$ are actually based on the observed beliefs of the receiver and the third party. The details can be seen in the following contents.

There have been a lot of methods that were used to identify the weight of principals. It is usually depended on the specified criteria. For example, three criteria '*past experiences*', '*background relevance*' and '*relevant credits*' are adopted for computing the weight of each principal. We need to synthetically consider the important ratios of the criteria and the evaluation results of the criteria for all principals. Nevertheless, it is not an emphasis to discuss how to calculate the weight of each principal here.

Until now, we have not referred closely to how the probability of the inference rules is specified. To authenticate a message m, the user has to check whether the held messages match at least one of the known rules. Sometimes, it is possible that there are more than one rule that can be satisfied.

Let $r_1, r_2, \cdots, r_n$ be a set of available inference rules. In general, suppose $Path_P$ represents a subset of rules that are used to authenticate the statement m.

Example 5.22. Consider a process of authentication shown in Figure 5.1. R can authenticate statement m by using two paths including $rule_1$ and $rule_2$. Nevertheless, M_T verifies m by using $rule_3$ only. More generally, we have

$$Path_R = \{rule_1,\ rule_2\} \text{ and } Path_T = \{rule_3\}.$$

If an authentication includes one path only, it is simple to compute the probability of the rule by using conditional probability. However, if there are multiple paths, it is necessary to use the probabilistic model to describe the probability of each way.

Let $\mathcal{S}$ be a set of sentences (such as the forms 'X *knows* m', 'X *knows* $Spb(Y)$' etc.) that the user considers as available knowledge for authentication. In particular, $\mathcal{S}$ may imply several inference rules, which can be used

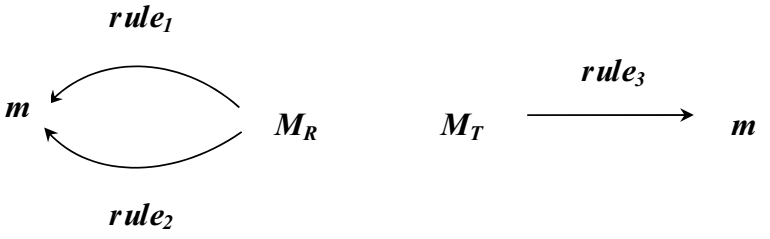

Fig. 5.1. Authentication Path

for the authentication. The sample space of the random experiment (i.e., the set of possible worlds considered by the user) is the power set of $\mathcal{S}$, denoted by $2^{\mathcal{S}}$. $P : 2^{\mathcal{S}} \rightarrow \mathbb{R}^+$ is a probability function on the sample space $2^{\mathcal{S}}$, which indicates a probability measure for all events (i.e., set of subsets of $\mathcal{S}$).

Suppose $A = \{A_1, \ldots, A_n\}$ denotes the conditions of $r : A_1 \times \ldots \times A_n \Rightarrow B$. As described above, the probability of a rule, namely p(r), is a conditional probability based on the conditions. The conditions are independent with each other since they represent independent sentences. We have

$$\begin{aligned} p(r) &= p(B|A_1 A_2 \cdots A_n) \\ &= p(AB)/p(A) \\ &= p(AB)/\prod_{i=1}^{n} p(A_i) \end{aligned}$$

where $p(A_i) > 0$ and $p(B)$ can be measured by using the support defined in Definition 5.6. Note, $p(r) = 0$ if $\exists\ p(A_i) = 0$.

Furthermore, we have $p(A) + p(B) - 1 \leq p(AB) \leq p(B)$ according to probability theorem. Consequently, we can obtain a probability interval regarding the rule. The minimum is selected as its probability to calculate the belief, namely

$$\begin{cases} if \prod_{i=1}^{n} p(A_i) \neq 0, & (p(A) + p(B) - 1)/\prod_{i=1}^{n} p(A_i) \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Example 5.23. Figure 5.2 presents an inference rules. It consists of conditions $pre_1, \cdots, pre_n$. They are represented and connected by using edges. The number above each edge indicates the user's support on each sentence pre_i.

On the other hand, there might be more than one path that the user can use to authenticate the specified statement. For example, in Figure 5.1, two rules including $rule_1$ and $rule_2$ can be applied to authenticate the statement m.

Definition 5.20. *Let r_i: $A_{i1} \times \ldots \times A_{in} \Rightarrow c_i$ and r_j: $A_{j1} \times \ldots \times A_{jk} \Rightarrow c_j$ be two inference rules, $i \neq j$. The r_i is said to be independent of r_j if* $\{A_{i1},$

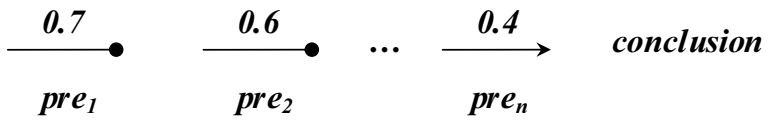

Fig. 5.2. The Assigned Values of Premises of Rule

$\ldots, A_{in}\} \cap \{A_{j1}, \ldots, A_{jk}\} = \emptyset$*; otherwise they are said to be dependent. We have*

(1) $p(r_i \cup r_j) = p(r_i) + p(r_j)$, if r_i is independent of r_j;
(2) $p(r_i \cup r_j) = p(r_i) + p(r_j) - p(r_i) * p(r_j)$, if r_i and r_j are dependent with each other.

Example 5.24. Suppose r_1: 'X knows m' × 'X knows Y's digital signature on m' × 'X knows Y's public signature key' ⇒ 'X authenticates Y, m' and r_2: 'X knows m' × 'X authenticates Y, $H(m)$' ⇒ 'X authenticates Y, m' are two rules, which can be used to authenticate the statement m. r_1 is not independent of r_2 because they have a common condition 'X knows m'.

Possibly, there are more than two rules used to authenticate a statement. In this extreme case, according to the inclusion-exclusion principle [106], the probability of the union of multiple rules can be computed by using the addition formula of probability theorem. Suppose n rules including $r_1, \cdots, r_n$ are used in the authentication. This gives

$$P(\bigcup_{i=1}^{n} r_i) = s_1 - s_2 + s_3 + \cdots + (-1)^{n+1} s_n$$

where $s_1 = \sum_{i=1}^{n} P(r_i)$, $s_2 = \sum_{1 \le i < j \le n} P(r_i r_j)$, $s_3 = \sum_{1 \le i < j < k \le n} P(r_i r_j r_k)$, $\cdots$, $s_n = P(r_i r_j \cdots r_n)$.

Although an authentication can include a number of rules in theory, we consider three rules only for the sake of brevity.

Suppose $\{r_{R1}, \cdots, r_{Rk}\}$ and $\{r_{T1}, \cdots, r_{Tl}\}$ are two set of rules used to validate m by the receiver R and the third party T, respectively. Therefore, the probability of rules in Definition 5.20 can be defined as.

$$p_R(r) = p(\bigcup_{i=1}^{k} r_{Ri}) \tag{5.4}$$

$$p_T(r) = p(\bigcup_{i=1}^{l} r_{Ti}) \tag{5.5}$$

Based on the obtained assumed belief, the observed belief and the probability of rules, we are then able to compute the trust that can be put on the goal of the protocol.

Merging inconsistent beliefs. It is observed that the principals might obtain inconsistent trust in the authenticated goal m. Certainly, this has a negative impact on evaluating the correctness of the protocol. Thus, such individual beliefs have to be merged to generate a commonly acceptable trust in the goal. In reality, it may be infeasible to require that all principals have high and consistent trust. Therefore, it is reasonable to use a tradeoff by majority criterion. In other words, a statement m is believe to be secure as long as the majority of the principals believe m is secure. We intend to merge the inconsistent beliefs according to the quantities of trust in the specified goal.

Refer to the strict weighted majority mentioned in [95], the combined weight of the support for m should be over 50% of the total weight. Let η be the threshold of weight. We have

$$\eta = \sum_{P \in \{S,R,T\}} \varpi_P / 2 \tag{5.6}$$

According to Definition 5.19, we can work out the sum of the belief of the sender, receiver and the third party with respect to m. We have

$$Bel(m) = Bel_S(m) + Bel_R(m) + Bel_T(m) \tag{5.7}$$

Based on the threshold η of weight and the sum of belief on the statement m, the user is able to determine whether m is secure or not by merging their inconsistent beliefs together.

$$\begin{cases} \text{if } Bel(m) \geq \eta, & m \textit{ is believed to secure by the user} \\ \textit{otherwise} & m \text{ is insecure} \end{cases}$$

The proposed merging operator provides a useful way to deal with inconsistent beliefs and complement the formal proof of security protocols.

5.3.4 Experiments

We now look at experiments of dealing with inconsistent belief in secure messages. For the reason of brevity, it assumes that all the messages transmitted between principals are fresh and the principals are trustworthy. Also, the messages are assumed to be generated and sent by the sender and received and seen by whom it claims to be. The sentences may express plaintext, encrypted messages, symmetric key, signature key or something like that. Two experiments are presented below to illustrate the proposed method.

Experiment 1. Suppose $M_S = \{\alpha, \beta, \phi, \theta, \alpha \wedge \beta \wedge \theta \rightarrow \gamma, \delta\}$, $M_R = \{\neg\alpha, \phi, \neg\gamma, \phi \wedge \neg\gamma \rightarrow \neg\theta, \neg\delta\}$, and $M_T = \{\alpha, \beta, \phi, \theta, \gamma, \delta\}$. Let $\alpha \equiv$ '*message* m', $\beta \equiv$ '*the sender's private signature key* $Spv(S)$', $\phi \equiv$ '*the sender's public signature key* $Spb(S)$', $\theta \equiv$ '*the sender's identity* ID_S', $\gamma \equiv$ '*the sender's digital*

signature $S(<ID_S, T, H(m)>, Spv(S))$' and $\delta =$ '*authenticated m*'. Let $\alpha' \equiv$ 'A knows α', $\phi' \equiv$ 'A knows ϕ', $\gamma' \equiv$ 'A knows γ' be statements, which are three conditions of the inference rule r: $\alpha' \times \phi' \times \gamma' \Rightarrow \delta$. Let the weight of S, R and T be $\varpi_S = \varpi_R = 0.6$ and $\varpi_T = 0.8$, respectively.

To generate the observed belief, we first need to transfer the messages known by the principals into models. According to the the definition of minimal QC model, we have $MQC(M_S) \equiv MQC(M_T) \equiv \{+\alpha, +\beta, +\theta, +\phi, +\gamma, +\delta\}$ and $MQC(M_R) \equiv \{\neg\alpha, +\phi, \neg\theta, \neg\gamma, \neg\delta\}$. Thus, we can calculate the support of the sentences. We just illustrate the authentication of α below and the authentication of remaining sentences is left to readers.

M_S, M_R and M_T all have one model only. Based on the Definition 5.17 and 5.18, we have $supp(M_S, \alpha) = supp(M_T, \alpha) = 1$, $supp(M_R, \alpha) = 0$, $supp(M_S, \phi) = supp(M_T, \phi) = supp(M_R, \phi) = 1$, $supp(M_S, \gamma) = supp(M_T, \gamma) = 1$, $supp(M_R, \gamma) = 0$, $supp(M_S, \delta) = supp(M_T, \delta) = 1$, $supp(M_R, \delta) = 0$. It is observed that S, R and T have inconsistent belief in α. In addition to the specified weight (assumed belief) and the support(observed belief), the probability of rules must be included to determine the final belief.

As a result, $Bel_S(m) = supp(M_S, \alpha) * \varpi_S = 0.6$ since the sender does not need to use inference rules to authenticate α. According to the formula (5.3), (5.4) and (5.5), we have

$$Bel_R(m) = \frac{supp(M_R,\alpha) * supp(M_R,\phi) * supp(M_R,\gamma) + supp(M_R,\delta) - 1}{supp(M_R,\alpha) * supp(M_R,\phi) * supp(M_R,\gamma)} * \varpi_R$$
$$= 0$$
$$Bel_T(m) = \frac{supp(M_T,\alpha) * supp(M_T,\beta) * supp(M_T,\gamma) + supp(M_T,\delta) - 1}{supp(M_T,\alpha) * supp(M_T,\beta) * supp(M_T,\gamma)} * \varpi_T$$
$$= 0.8$$

Thus, we obtain $Bel(m) = Bel_S(m) + Bel_R(m) + Bel_T(m) = 1.4$ in terms of the formula (5.7).

From the formula (6.6), the 50 percent of the weight $\eta = (0.6 + 0.6 + 0.8)/2 = 1 < Bel(m)$. Therefore, m is believed to be secure in this transaction using the protocol.

Experiment 2. Suppose the meaning of α, β, γ, θ, ϕ, α', β' and γ' keep the same as the **Experiment 1**. $M_S = \{\alpha, \neg\alpha, \beta, \varphi, \phi, \theta, \alpha \wedge \beta \wedge \theta \rightarrow \gamma, \delta\}$, $M_R = \{\neg\alpha, \alpha, \phi, \varphi, \neg\gamma, \phi \wedge \neg\gamma \rightarrow \neg\theta, \neg\delta\}$ and $M_T = \{\alpha, \beta, \phi, \varphi, \theta, \gamma, \delta\}$. Let $\varphi \equiv$ '*the hashing of message m*' and $\varphi' \equiv$ '*A authenticates* φ'. *Let* $\varpi_S = \varpi_R = 0.4$ and $\varpi_T = 0.9$.

There are two rules that can be used to authenticate α in this experiment. The user can use not only the rule mentioned in the last experiment r_1: $\alpha' \times \beta' \times \gamma' \Rightarrow$ 'A authenticates α', but also another rule r_2: $\alpha' \times \varphi' \Rightarrow$ 'A authenticates α'. It is noted that there is an intersection between the conditions $\{\alpha', \beta', \gamma'\}$ of r_1 and the conditions $\{\alpha', \varphi'\}$ of r_2, namely $\{\alpha', \beta',$

$\gamma^{'}\} \cap \{\alpha^{'}, \varphi^{'}\} = \alpha^{'}$. On the other hand, the weights of sender and receiver are down but the weight of the third party is up. Thus, more of the third party's opinion will be reflected in determining the final belief in m. Actually, it is usually reasonable to put more trust on the third party like trust center or authority.

Similarly, we need to first generate the models of M_S, M_R and M_T. In the same way, we can obtain $MQC(M_S) \equiv \{+\alpha, \neg\alpha, +\varphi, +\beta, +\theta, +\phi, +\gamma, +\delta\}$, $MQC(M_T) \equiv \{+\alpha, +\beta, +\varphi, +\theta, +\phi, +\gamma, +\delta\}$ and $MQC(M_R) \equiv \{+\alpha, \neg\alpha, +\varphi, +\phi, \neg\theta, \neg\gamma, \neg\delta\}$.

Thus, according to the formula (5.1), we have $supp(M_S, \alpha) = supp(M_R, \alpha) = 0.5$, $supp(M_T, \alpha) = 1$, $supp(M_S, \beta) = supp(M_T, \beta) = 1$, $supp(M_R, \beta) = 0$, $supp(M_S, \gamma) = supp(M_T, \gamma) = 1$, $supp(M_R, \gamma) = 0$, $supp(M_S, \delta) = supp(M_T, \delta) = 1$, $supp(M_R, \delta) = 0$, $supp(M_S, \varphi) = supp(M_T, \varphi) = supp(M_R, \varphi) = 1$. Unlike the **Experiment 1**, the supports for α vary in M_S and M_R since they contain $\neg\alpha$. Using the obtained support, we are able to calculate the probabilities of rules.

In the similar manner, we have $Bel_S(m) = supp(M_S, \alpha) * \varpi_S = 0.5 * 0.4 = 0.2$. There are two rules that can be used to authenticate α in this experiment. According to the formula (5.3), (5.4) and (5.5), we have

$$
\begin{aligned}
p_R(r_1) &= \frac{supp(M_R,\alpha) * supp(M_R,\beta) * supp(M_R,\gamma) + supp(M_R,\delta) - 1}{supp(M_R,\alpha) * supp(M_R,\beta) * supp(M_R,\gamma)} \\
&= 0 \\
p_R(r_2) &= \frac{supp(M_R,\alpha) * supp(M_R,\varphi) + supp(M_R,\delta) - 1}{supp(M_R,\alpha) * supp(M_R,\varphi)} \\
&= 1 \\
p_T(r_1) &= \frac{supp(M_T,\alpha) * supp(M_T,\beta) * supp(M_T,\gamma) + supp(M_T,\delta) - 1}{supp(M_T,\alpha) * supp(M_T,\beta) * supp(M_T,\gamma)} \\
&= 1 \\
p_T(r_2) &= \frac{supp(M_T,\alpha) * supp(M_T,\varphi) + supp(M_T,\delta) - 1}{supp(M_T,\alpha) * supp(M_T,\varphi)} \\
&= 1
\end{aligned}
$$

Thus, $p_R(\mathrm{r}) = p_R(r_1) + p_R(r_2) - p_R(r_1) * p_R(r_2) = 0 + 1 - 0 = 1$ and $p_T(\mathrm{r}) = p_T(r_1) + p_T(r_2) - p_T(r_1) * p_T(r_2) = 1 + 1 - 1 = 1$

We have $Bel_R(m) = p_R(\mathrm{r}) * \varpi_R = 0.4$ and $Bel_T(m) = p_T(\mathrm{r}) * \varpi_T = 0.9$, and $Bel(m) = 0.2 + 0.4 + 0.9 = 1.5 > \eta = (0.4 + 0.4 + 0.9)/2 = 0.85$. As a result, m is believed to be secure in this transaction using the protocol.

5.4 Summary

Traditional formal analysis for security protocols usually assumes that the principals are trustworthy and the communication channels are safe and reli-

able. They have been successful in detecting security flaws and ambiguity in the protocols. However, in electronic transactions, the messages are exchanged between principals under a hostile/uncertain environment. This may result in inconsistent secure messages or conflicting beliefs between principals. Their estimation is viewed as uncertainty issues in secure messages.

The inconsistency in secure messages has been a significant challenge to the reliability of verification results and made the e-commerce activities at risk. Although there have been considerable efforts in handling inconsistency issues, they however focus on inconsistency between knowledge bases and do not provide a numerical estimation to the inconsistency. Unlike the information from traditional knowledge base, the features of secure messages must be considered to measure the inconsistency in secure messages correctly.

In this chapter, we first developed a formal framework to deal with the inconsistency in secure messages, taking into account the properties of freshness and dynamics of secure messages. The objective is to use this logical framework in context of the formal analysis of security protocols. It achieves:

- measuring the inconsistency in secure messages with weight that represents the degree of importance of message sources; and
- analysing the inconsistent secure messages by evaluating their reliability.

It enables the identification of uncertain messages from the secure and insecure messages. We have illustrated the use of this framework with some examples, and evaluated the proposed approach experimentally. The experimental results demonstrate that this method is useful to ensure the reliability of the trust on the goal that can be put on the protocol.

Moreover, we proposed a probabilistic method to intuitively measure the inconsistent beliefs between principals, and intended to handle the inconsistent beliefs by a weighted majority criterion. In particular, the freshness and dynamics properties of secure messages are taken in to account. This is able to complement and enhance the formal proof of correctness of the protocols.

The belief in secure messages is classified into three categories, including assumed belief, observed belief and the inferred belief. The combination of the above beliefs represents the transmission of belief from initial belief to new belief. In particular, this chapter presents a numeric estimation to measure the inconsistent beliefs by using the MQC models. Moreover, we use minimal conditional probability of rules and take into account dependent rules when calculating the probability of rules.

The presented experiments demonstrate that our method can effectively handle the inconsistent beliefs in secure messages, intuitively measure the trust in the goal that can be put on the protocol, and ensure the correctness of the proof of security protocols.

6

Applications of Data Mining in Protocol Analysis

Traditional formal approaches such as theorem proving and model checking have been widely used to analyse security protocols. Ideally, they assume that the data communication is reliable and require the user to predetermine authentication goals mentioned in Chapter 5. However, missing and inconsistent data have been greatly ignored, and the increasingly complicated security protocols make it difficult to predefine such goals. We thus presents a novel approach in this chapter to analyse security protocols using association rule mining. It is able to not only validate the reliability of transactions but also discover potential correlations between secure messages. The algorithms and conducted experiments demonstrate that our approaches are useful in enhancing the current protocol analysis.

The rest of this chapter is organized as follows. We start from Section 6.1 by introducing inconsistent secure messages and data mining. Section 6.2 briefly overviews the related work. Section 6.3 presents some basic concepts. Section 6.4 presents how to analyse inconsistent secure messages using association rule mining. algorithms and experiments are described in Section 6.5. Finally, we conclude this chapter in Section 6.6.

6.1 Introduction

The rapid growth of electronic commerce (e-commerce) not only plays a non-trivial role in global economy but also poses a big challenge to security in e-commerce. A plenty of security protocols have been developed to ensure data integrity and confidentiality [38, 74, 140] in e-commerce. However, the design of security protocols is usually error-prone [22], and missing and spurious data are prevalent in data gathering today.

Traditional formal methods to analysing security protocols are mainly classified by theorem proving [22, 44, 120] and model checking [68]. They have been successful in modelling the behaviour of a protocol and mathematically

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 175–192, 2008.

verifying that the protocol design and implementation satisfy safety requirements of protocols. Usually, the verification starts from the assumption, via intermediate formulae, to the authentication goal predefined by users [44]. In other words, users should have clear ideas about what the suspectable problems are. Nevertheless, it becomes very hard or even infeasible in case of a complicated security protocol such as financial transaction protocol. Also, some hidden or ambiguous security problems are not easy to detect by users.

Traditional formal analysis for security protocol unrealistically assumes that the data communication is secure. However, the transmitted secure messages in a hostile environment can result in missing or inconsistent values such as a user's identity, credit card number and password in transaction databases. Furthermore, these inconsistent messages are in fact interactional. For example, a tampered user's password may imply the potential divulgence of the user's credit card number and identity. Therefore, there is an urgent need to improve and enhance the analysis of increasingly complicated security protocols by identifying the potential correlations between secure messages.

In the past few years, data mining techniques emerged as a means of identifying patterns and trends from large quantities of data [3, 36, 67]. Among them, association rule mining is a popular summarization and pattern extraction algorithm to identify the correlations between items in transaction databases [163]. Unlike the conventional association rule discovery, we extend the original idea to association rule mining of inconsistent secure messages [32]. In order to conform to the new idea, we need to make considerable extension to the original setting. It is briefly summarized as:

- Firstly, the missing messages in itemsets should be taken into account. For example, given $A' = \{expiration\ date, password_2, account\ number\}$, it misses message *name*. Therefore, itemset $A = \{expiration\ date, password_1, account\ number, name\}$ and A' are regarded as two different itemsets, namely $A' \neq A$.
- Even all messages in two itemsets are corresponding, they must be consistent, and otherwise they are deemed to be inconsistent itemsets. For example, since the $password_2$ in A' is not equal to $password_1$ in A, they are inconsistent.

During transactions, if the messages in itemset A were lost or tampered, this will lead to the decrease of the number of occurrence of this itemset, and in this way, the degree of support and confidence for related rules will decrease either. And thereby, it is reasonable to say this transaction is insecure in case that the support and confidence of the rule is smaller than the specified *minimum support* and *minimum confidence* by users or experts.

This chapter presents how to use association rule mining to analyse security protocols and identify the potential correlations between secure messages. Unlike traditional market basket data, the freshness of secure messages and

validity of public keys must be examined before extracting frequent itemsets. In particular, missing and inconsistent items in transaction databases are converted to operable data for further data mining.

6.2 Related Work

Security protocols have become the requisite of e-commerce systems. However, they easily suffer from malicious attacks due to their ambiguous specifications, and their designs are a difficult and error-prone task [65]. A variety of methods and tools have been developed to verify these protocols [112]. Among them, theorem proving and model checking have gained many attentions.

As to theorem proving, BAN logic [22] is the representative work among them, from which a number of approaches are developed [62]. It expresses the assumption and goal as statements in a symbolic notation so that the logic can proceed from a known state to one where it can ascertain whether the goal is in fact reached. Therefore, it is not surprising that the predetermination of authentication goals can become difficult owing to increasingly complicated security protocols. Additionally, some latent flaws are not easy to detect. Model checking is another kind of approaches aiming at automated verification of security protocols. Lowe [97] used Failures Divergences Refinement Checker (FDR) to debug and validate the correctness of Needham-Schroeder protocol and Heintze [68] used FDR to verify NetBill and a simplified digital cash protocol. However, they ideally assume that the communication channel and principal are secure and trustworthy. The inconsistency between secure messages is partially neglected.

There have been many approaches in tackling the inconsistency in knowledge bases, such as arbitration based information merging [93] and majority based information merging [96]. Nevertheless, they focus on the handling of incoherence in knowledge base rather than the inconsistency in secure messages. A logical framework to merging inconsistent secure messages was presented in [4]. Chapter 5 presents approaches to measure the inconsistency in secure messages and the conflicting belief in secure messages. However, none of them is able to identify potential associations between secure messages.

Data mining, with its potential to discover hidden and valuable information from large databases has been successfully used to identify patterns and trends from large quantities of data [3, 67]. Data mining has been applied in a variety of areas, such as financial data mining, text mining, data mining in healthcare and data mining in bioinformatics. Among them, association rule mining plays an important role in identifying correlations between items in transaction databases [163]. It is used to discover elements that co-occur frequently within a data set, and to identify rules by reducing a potentially huge amount of information to a small, understandable set of statistically supported statements.

Detection of security threats. Recently there has been a preconization that data mining can be used for security purposes. One application is the use of data mining to improve security, such as intrusion detection [91]. A second application is the potential security hazards posed when an adversary has data mining capabilities [4]. Data mining can be a potential means to detect credit card fraud by analysing massive amounts of transaction data in a timely manner; and identify and track individual activities such as money transfer and communications for homeland security. For example, (1) an investigation said a stolen credit card often is used to make a self-service purchase at a gas station (to determine if the card is still active) immediately before it's used to buy jewelry or for some other major purchase. Such illicit transaction patterns really stand out when the system has been 'trained' to recognize the legitimate cardholder's usage pattern. An irregular transaction prompts an alert, which is transmitted instantly to the sales clerk handling the purchase; (2) to identify potential terrorist suspects in a large pool of individuals, the user may test the model using data that includes information about known terrorists.

The detection models in [91] aim to develop a systematic framework to semi-automate the process of building intrusion detection systems rather than traditional intrusion systems constructed by manual and ad hoc means. Audit mechanisms are needed to record system events, distinct evidence of legitimate and intrusive activities will be then manifested in the audit data. Thus, a normally infrequent failure may be easily detected in case of a large number of consecutive failures due to intrusions. Its basic idea is to first compute the association rules and frequent episodes from audit data. These patterns are then used, with user participation, to guide the data gathering and features selection processes.

The identification of frequent episodes is based on minimal occurrences. Suppose there is an event database $\mathcal{D}$, in which each transaction is associated with a timestamp, an interval $[t_1, t_2]$ is the sequence of transactions that starts from t_1 and ends at t_2. Let X be an itemset in $\mathcal{D}$. An interval is a minimal occurrences of X if it contains X and none of its sub-intervals contains X.

- $support(X)$ is the number of minimal occurrences, which contains itemset X and the number of records in $\mathcal{D}$
- A frequent episode is the form of X, Y, Z, c, s, *window*, where X, Y and Z are itemsets in $\mathcal{D}$, $s = support(X \cup Y \cup Z)$ is the support of the rule, and $c = support(X \cup Y \cup Z)/support(X \cup Y)$ is the confidence. The width of each of the occurrences must be less than *window*.

Moreover, the authors adapt previous mining algorithms to reduce irrelevant rules or discover the low frequency patterns. The discover patterns will be used as the indicator for gathering data and as the basis for selecting appropriate temporal statistical features. Two rules r_1 and r_2 can be merged into one rule

if 1) their right and left hand sides are exactly the same; and 2) the support values and the confidence values are close within a defined threshold. For example, (*service* = ***smtp***, *src bytes* = 100) and (*service* = *smtp*, *src bytes* = 200) can be combined to (*service* = *smtp*, 200 ≤ *src bytes* ≤ 300). The merged rule set can indicate whether the audit data has covered sufficient variations of behaviour.

Protection of privacy. Although data mining techniques have recently touched on privacy issues, they put emphasis on how to protect sensitive knowledge before sharing. For example, a two-party algorithm [5] is presented to efficiently discover frequent itemsets with minimum support levels, without either revealing individual transaction values. Stanley and Osmar proposed a new framework for enforcing privacy in mining frequent itemsets, in which it combines techniques for efficiently hiding restrictive patterns [142].

As described in [4], the privacy concerns in data mining demand us to develop accurate models without access to precise information in individual data records. The authors thus build a decision-tree classifier from training data in which the values of individual records have been perturbed. The resulting data records are very different from the original records. A novel reconstruction procedure is then proposed to accurately estimate the distribution of original data values.

This method focuses on two value distortion methods, in which a value $x_i + r$ is returned instead of x_i where r is a random value drawn from two distributions.

- **Uniform**: The random variable has a uniform distribution, between [-α, +α]. The mean of the random variable is 0.
- **Gaussian**: The random variable has a normal distribution, with mean $\mu = 0$ and standard deviation σ.

A measure is established to qualify privacy in terms of how closely the original values of a modified attribute can be estimated. It is estimated by using c% confidence that a value x lies in the interval [x_1, x_2] and the interval width (x_2 - x_1) that defines the amount of privacy at c% confidence level.

It is necessary to reconstruct the original data distribution from the randomized data. Suppose $x_1, x_2, \cdots, x_n$ are n original values as realization of n independent identically distributed (iid) random variables $X_1, X_2, \cdots, X_n$, each with the same distribution as the random variable X. To hide their values, n independent random variables $Y_1, Y_2, \cdots, Y_n$ have been used, each with the same distribution as a different random variable Y. Given $x_1 + y_1$, $x_2 + y_2, \cdots, x_n + y_n$ and the cumulative distribution function F_Y for Y, it aims to estimate the cumulative distribution function F_X for X.

Another limitation of current data mining is that while it can identify connections between behaviors and/or variables, it does not consider the properties of secure messages. For example, an application may identify that prin-

cipal A's public/private key is related to a collusion attack. However, the public key is usually regarded as a different item from the private key. In fact, it is necessary to know both keys to generate A's signature. Thus, the public key and private key of principals are viewed as identical items. Moreover, the freshness and validity of secure messages must be verified before data mining. This is to ensure that the obtained transaction data is not a fraud. Unfortunately, there is a lack of work that is closely related to security protocol analysis using data mining.

6.3 Basic Concepts

Cryptography [130] is an essential tool to achieve data security such as authentication, integrity and confidentiality in e-commerce systems. In general, it is classified into asymmetric cryptography and symmetric cryptography. Symmetric cryptography uses the same key (the secret key) to encrypt and decrypt a message, and asymmetric cryptography use one key (the public key) to encrypt a message and another key (the private key) to decrypt it. The more details regarding cryptography principles can be seen in Chapter 1

Example 6.1. Alice can use a shared symmetric key k to encrypt a letter and send it to *Bob*. *Bob* can use the same key to decrypt it. If *Alice* uses her private key $K^{-1}(Alice)$ to sign the letter, *Bob* uses *Alice*'s public key $K_p(Alice)$ to open it and knows it was really signed by *Alice*.

In addition to the cryptographic strength of cryptography algorithms, the security of e-commerce systems largely depends on the reliability of security protocols that cover the full range of administrative and technical measures that need to fulfill corporate security objectives. The following two fundamental elements are often highlighted in the formal analysis of security protocols:

- replay of messages that presents a message from different context is used into the intended context to fool the honest participant into thinking they have successfully completed the protocol run; and
- correct correlation of cryptography keys with specified principals.

The timestamp has been proved to be an efficient way to ensure the freshness of secure messages in [40]. It plays an important role in preventing the replays of former transmitted secure messages. The basic idea is to integrate a timestamp with the delivered message, and then check its validity by measuring the discrepancy between server clock and local clock and the expected network delay time (for received messages), or comparing it with the expiration time (for generated messages). The follows present an instance and the details can be seen in Chapter 5

Example 6.2. If the attached timestamp T is *30 Nov 2004 18:35:20 +1000*, and the specified expiration time $T_{expiration}$ is *30 Nov 2004 18:34:28 +1000*, we can say the message is not fresh due to $T > T_{expiration}$.

Authentication can be further enhanced by using certificates that are digitally signed by recognized certificate authorities (CA) and are used to specify the affiliation between public keys and principals. Figure 3.2 describes a public-key infrastructure (PKI) that provides the issue, management and use of public keys and certificates for authentication, privacy and other security properties. A certificate is verified following the trust tree to a known trusted party.

In e-commerce systems, the authentication may occur in different transactions and different places. According to the source of transaction data, secure messages are classified into three categories:

- D_S represents the transaction database where messages are generated and sent to the receiver;
- D_R represents the transaction database where messages are received and authenticated; and
- $D_T = \{D_{T_1}, \cdots, D_{T_m}\}$ $(1 \leq m)$ represents all relevant transaction databases to the third party such as financial institutions and certificate issuers.

The transmitted secure messages stored in the above transaction databases are actually valuable data sets that can be used to check the correctness of security protocols using data mining. However, unlike the traditional market basket data, the inherent properties of secure messages such as freshness require us to perform additional authentication of their validity before starting data mining. Correspondingly, a supporting relationship $\models$ is needed to authenticate the secure messages.

(1) $\models_D \alpha$ iff $\alpha \in D$, and α is fresh;
(2) $\models_D K_p(X)$ iff $K_p(X) \in D$, and $K_p(X)$ is authenticated to be valid.

Here α indicates a secure message, $D \in \{D_S, D_R, D_T\}$, and X represents the principal who sent messages to the receiver.

However, there may exist conflicting supports between different transaction databases. For example, $\models_{D_S} \alpha$ and $\models_{D_R} \neg\alpha$ show discrepant supports between D_S and D_R. This phenomenon in fact indicates the possibility of potential security flaws in security protocols.

As mentioned above, the user must ensure that the security requirements are satisfied to authenticate a message. In other words, they must all be true conditions to achieve the expected goal. As a result, the authentication of each transaction can be viewed as an association rule including delivered secure messages between principals. The trust in the authenticated goal can be measured by the traditional support-confidence framework of data mining [3].

The more messages (conditions) of the rule are tampered, the less frequent the itemset will be. Therefore, the reliability of a transaction (or the trust in the goal) can be evaluated in virtue of the support and confidence of the corresponding association rule.

6.4 Association Rule Mining for Inconsistent Secure Messages

This section adapts and extends the original association rule mining to analyse inconsistent secure messages and identify the potential correlations between secure messages.

6.4.1 The Basics of Association Rule Mining

Let $I = i_1, \cdots, i_n$ be a set of items and D be a collection of transactions, called transaction database. Each transaction $T \in D$ consists of a collection of items. Let $A \subseteq I$ be an itemset. We can say that a transaction T contains A in case $A \subseteq T$. An itemset A in a transaction database D has a support, denoted as $supp(A)$. Hence we have

$$supp(A) = |T_A|/|D|\%$$

where T_A represents transactions in D, which contain the itemset A.

An itemset A in D is called a frequent itemset if its support is equal to, or greater than, a given frequency threshold λ (*minimum support*) by user or experts, namely $supp(A) \geq \lambda$. An association rule is an implication of the form, $A \rightarrow B$, where A and B are frequent itemsets, and $A \cap B = \emptyset$. For each association rule, we can use the support-confidence framwork [3] to measure it. A rule $A \rightarrow B$ is valid if

(1) $supp(A \cup B) \geq minsupp$
(2) $conf(A \cup B) = supp(A \cup B)/supp(A) \geq minconf$

where *minsupp* and *minconf* are designated by users or experts. Note that we do not discuss how to select optimal minimum support and minimum confidence in this book. Association rule provides a simple but efficient form of rule patterns for data mining. It is briefly summarized as follows:

- Firstly, we need to extract all frequent itemsets from the transaction database.
- Secondly, we determine valid rules from discovered frequent itemsets according to the support and confidence of the rules.

Table 6.1. A transaction database

PID	*Items*				
P_1	m_1		m_3	m_4	
P_2		m_2	m_3		m_5
P_3	m_1	m_2	m_3		m_5
P_4		m_2			m_5

To illustrate the use of the support-confidence framework, we present an example of mining association rules below:

In Table 6.1, the universe $I = \{m_1, m_2, m_3, m_4, m_5\}$. Each row in the table can be viewed as a transaction database of a principal. For example, m_1 = *order*, m_2 = *encryption key* k^{-1}, m_3 = *encrypted order using* k^{-1}, m_4 = *decryption key* k, m_5 = *order after decryption*.

Let *minsupp* = 50% and *minconf* = 60%. According to the support-confidence framework, we show a simple mining process in two steps:

(1) The mining starts from calculating the frequencies of k-itemsets. It is observed that m_1 occurs in P_1 and P_3, so its frequency is 50%, which is equal to the *minsupp*. m_2 occurs in P_2, P_3 and P_4, and its frequency is 75%, which is greater than *minsupp*. The frequencies of m_3, m_4 and m_5 are 75%, 25% and 75%, respectively. As a result, m_1, m_2, m_3 and m_5 are frequent 1-itemsets. Furthermore, $\{m_1, m_3\}$ occurs in P_1 and P_3, its frequency is 50%, which is equal to *minsupp*; $\{m_1, m_2\}$ occurs in P_3 only, its frequency is 25%, which is less than *minsupp*. In the similar way, we can determine remaining frequent 1-itemsets, 2-itemsets and 3-itemsets that are presented in Table 6.2, Table 6.3 and Table 6.4, respectively. Note that there is no frequent 4-itemsets in this case.

Table 6.2. Frequent 1-itemsets in the database

Frequent itemsets	*Frequency*
$\{m_1\}$	50%
$\{m_2\}$	75%
$\{m_3\}$	75%
$\{m_5\}$	75%

(2) The second step aims to identify association rules from the obtained frequent itemsets. Based on the Table 6.2, Table 6.3 and Table 6.4, $\{m_2, m_3, m_5\}$, $\{m_2, m_3\}$ and $\{m_5\}$ are frequent 2-itemsets and 3-itemset and 1-itemset, respectively. Furthermore, we have $supp(\{m_2 \cup m_3 \cup m_5\})/supp(\{m_2 \cup m_3\}) = 1$, which is greater than *minconf* = 60%. Thus, $m_2 \cup m_3 \rightarrow m_5$ can be identified as a valid rule. This means that if there is

Table 6.3. Frequent 1-itemsets in the database

Frequent itemsets	*Frequency*
$\{m_1, m_3\}$	50%
$\{m_2, m_3\}$	50%
$\{m_2, m_5\}$	75%
$\{m_3, m_5\}$	50%

Table 6.4. Frequent 3-itemsets in the database

Frequent itemsets	*Frequency*
$\{m_2, m_3, m_5\}$	50%

a higher support to the *encryption key* and *encrypted order* in the transaction database, the *order after decryption* is believed to be secure. In this regard, it demonstrates the trust in the goal that can be put on the protocol can be transferred to mining association rules.

There have been a great many efforts to develop algorithms or tools to increase the efficiency of identifying the frequent itemsets and association rules [36, 67, 163]. Nevertheless, it is not an emphasis to discuss the topic of data mining in this book. The details can be seen from relevant literatures.

6.4.2 Data Preparation

Traditional association rule mining does not consider the missing and inconsistent data. If an item is missed in a transaction, it will simply not be counted when calculating the frequency of related itemsets. In particular, if an item is inconsistent with another item, they will be viewed as different items, whereas they may be correlated, such as $k \neq k'$ (tampered k by an intruder). These will result in incorrect computation of the frequency of itemsets, and finding unexpected patterns. Thus, it requires us to standardize the obtained transaction data from principals to ensure correct data mining.

Unlike the market basket data, secure messages have some properties, as mentioned above. In addition, the local support from individual transaction databases needs to be integrated to obtain a global support from databases of all participants. Therefore, it is necessary for us to extend the previous association rule mining to handle inconsistent secure messages.

The secure messages in transaction databases can be viewed as items. Suppose $D = \{D_1, \cdots, D_k\}$ is a set of transaction databases. Each transaction database consists of a collection of secure messages. Let $I = \{x \mid x \in D_i, 1 \leq i \leq k\}$ be a set of items. $A \subseteq I$ and $B \subseteq I$ are itemsets. An association rule is an implication of the form $A \rightarrow B$ where $A \cap B = \emptyset$. The rule $A \rightarrow B$ has

support, s in the set of transaction databases if $s\%$ of transaction databases contains $A \cup B$. The association rule has confidence, c in the set of transaction databases if $c\%$ of transaction databases containing A contains $A \cup B$.

Example 6.3. To register an account, the cardholder needs to fill out the registration form from *CA* with information such as the *cardholder*'s *name*, *date of birth*, *expiration date* and *account billing address*. Let $I = \{$*cardholder*'s *name*, *date of birth*, *expiration date*, *account billing address*$\}$ be the set of items. Hence we can say $\{$*cardholder*'s *name*, *date of birth*$\}$ and $\{$*expiration date*, *account billing address*$\}$ are itemsets as usual.

Suppose that a transaction $T = D_1 \cup D_2 \cup \cdots \cup D_n$ comprises n transaction databases, in which D_i $(1 \leq i \leq n)$ may be a sender, a receiver or the third party. If D_i contains φ and believes its freshness or validity when φ is a public key, itemset φ has local support from D_i, namely $\models_{D_i} \varphi$. The global support of itemset φ actually integrates the local support from all transaction databases in T. Additionally, the missing item is assigned *null* value, and the inconsistent item is denoted by using symbol $\neg$.

For simplicity, all itemsets in the rest of this chapter are assumed to be valid (freshness of messages or validity of keys) in the corresponding transaction databases. The relevant authentication of secure messages can be seen in Chapter 3 and Chapter 5.

Definition 6.1. *Let D_i be a transaction database and I be a set of items. Suppose $\phi = \{m_1, m_2, \cdots, m_k\} \subseteq I$ is an itemset. Then,*

$$\models_{D_i} \phi \text{ iff } \models_{D_i} \forall\ m_i \in \phi$$

Example 6.4. Let ID be an identity number and AC be an account number. Table 6.5 indicates the secure messages derived from the transaction database of different principals. The item account number AC is contained in D_1, D_2 and D_4 but missed in D_3. Thus, $\models_{D_1} AC$, $\models_{D_2} AC$, $\models_{D_4} AC$ and $\models_{D_3} \neg AC$. It is observed that AC and ID are missing in D_3 and D_4 respectively. In addition, D_3 contains an inconsistent item $\neg ID$. Consequently, we have $\models_{D_1} AC \cup ID$, $\models_{D_2} AC \cup ID$ due to $\models_{D_1} ID$ and $\models_{D_2} ID$, whereas, $\not\models_{D_2} AC \cup ID$ and $\not\models_{D_2} AC \cup ID$ due to $\not\models_{D_3} AC$, $\models_{D_3} \neg ID$ and $\not\models_{D_4} ID$.

From the observation, secure messages are different from traditional market basket data owing to their security properties. Although we ideally assume that secure messages are valid and there is correct association between the public key and principal, this cannot exclude the possibility of inconsistency, including missing and inconsistent values, between secure messages due to the potential *message loss*, *communication block* and *broken cipher*. Moreover, the inconsistency has an effect on measuring the trust in the goal of the protocol as mentioned in Chapter 5.

In particular, keys are different from ordinary secure messages. For example, in public-key cryptography, each principal has a pair of keys: a public key

Table 6.5. Secure Message Sources.

Database	Account number	Identifier	Key
D_1	AC	ID	$K^{-1}(X)$
D_2	AC	ID	$K_p(X)$
D_3	*null*	$\neg ID$	$K_p(X)$
D_4	AC	*null*	$\neg K_p(X)$

and a private key. Usually, the public key is known to everybody but the private key is known by the sender of the message only. Nobody can forge his/her signature without knowledge of his/her private key. Therefore, for each public/private key pairing, *private key* and *public key* are viewed as identical items when computing the corresponding support and confidence. The traditional association rule mining should be extended to deal with the public/private key pairs and address the inconsistency between secure messages.

(1) *Missing Item.* If an itemset misses some items, it will be viewed as an inconsistent itemset from the original one. For example, in Table 6.5, $\{AC, ID\}$ in D_1 and D_2, and $\{AC, null\}$ in D_4 are viewed as two inconsistent itemsets since the item ID is missed in D_4.
(2) *Tampered Item.* If some items are tampered in an itemset, the itemset is regarded as an inconsistent itemset with the original one. For example, in D_1, the item $\neg ID$ in D_3 is a tampered item in contrast to item ID in D_1 and D_2. And thereby, itemset $\{null, \neg ID\}$ in D_3 and itemset $\{AC, \neg ID\}$ in D_1 and D_2 are viewed as inconsistent itemsets.
(3) *Itemsets with Public/Private Key Pairs.* For any two itemsets, an itemset contains private key and the other contains public key. If the *public key* and *private key* are correctly matched pairs and the encrypted item are identical, they are regarded as identical itemsets. For example, in Table 6.5, the public key $K_p(X)$ in D_2 and D_3 matches with the private key $K^{-1}(X)$ in D_1 correctly. Nevertheless, $\neg K_p(X)$ in D_4 is a tampered public key. Therefore, itemset $\{AC, K^{-1}(X)\}$ in D_1 and itemset $\{AC, K_p(X)\}$ in D_2 are viewed as identical itemsets, whereas, itemset $\{AC, K_p(X)\}$ in D_2 and itemset $\{AC, \neg K_p(X)\}$ in D_4 are viewed as inconsistent itemsets owing to the inconsistency between $K_p(X)$ and $\neg K_p(X)$.

6.4.3 Identifying Association Rules of Interest

During a transaction, if secure messages in an itemset are lost or tampered, or public/private key pairs are not correctly matched, this will result in the decrease of the number of occurrences of this itemset. Consequently, the degree of support and confidence on the relevant association rule will decrease.

Therefore, if the rule is not of interest, it is reasonable to say the corresponding transaction is insecure. In order to compute the support of itemsets, we need to extend the original association rule mining.

Definition 6.2. *Suppose D_i, $1 \leq i \leq n$, is a transaction database in transaction T. Let φ be an itemset and $\varphi(D_i) = \{D_i \text{ in } T \mid D_i \text{ contains } \varphi\}$. Let the global support of φ be $supp(\varphi)$.*

$$supp(\varphi) = \sum_{i=1}^{n} |\varphi(D_i)|/|T| \tag{6.1}$$

Example 6.5. In Table 6.5, item AC is contained in D_1, D_2 and D_4 but not in D_3. Consequently, we have $|AC(D_1)| = 1$, $|AC(D_2)| = 1$, $|AC(D_4)| = 1$ and $|AC(D_3| = 0$. In the same way, we have $|\{AC \cup ID\}(D_1)| = 1$, $|\{AC \cup ID\}(D_2)| = 1$, $|\{AC \cup ID\}(D_4)| = 0$ and $|\{AC \cup ID\}(D_3)| = 0$ because AC and ID are missing in D_3 and D_4, respectively. Therefore, $supp(AC) = 3/4 = 0.75$ and $supp(AC \cup ID) = 2/4 = 0.5$.

An association rule is the implication χ: $A \rightarrow B$, where $A \cap B = \emptyset$. Therefore, the confidence of the rule $A \rightarrow B$ is

$$conf(A \rightarrow B) = \frac{\sum_{i=1}^{n} |\chi(D_i)|/|T|}{supp(A)} \tag{6.2}$$

Example 6.6. In an online booking, the user needs to fill out a form with *credit card number*, key *k*, *amount* and *address*. They are encrypted and sent to merchant *Y*. Initially, *Y* needs to authenticate the received message via the third party such as financial institutions. Suppose $D_1 = \{card\_number, k, amount, address\}$, $D_2 = \{card\_number, k, amount\}$ and $D_3 = \{card\_number, k, address\}$ are transaction databases. Let $minsupp = 50\%$ and $minconf = 60\%$. We have $supp(\{card\_number, k, amount\}) = 2/3 > 0.5$, $supp(\{card\_number, k\}) = 1 > 0.5$ and $conf(\{card\_number, k\}) \rightarrow \{amount\}) = 2/3 >$ o.6. $\{card\_number, k\} \rightarrow \{amount\}$ can be extracted as a valid rule.

The derived rule indicates that this transaction is secure and the number of lost and tampered messages is acceptably low. On the contrary, if the inconsistency between secure messages is low, the support and confidence of association rules will turn to be high. Hence we have

1. $belief(A \rightarrow B) =$ "*secure*", if $supp(A \cup B) \geq minsupp$, $conf(A \rightarrow \mathrm{B}) \geq minconf$;
2. $belief(A \rightarrow B) =$ "*insecure*", otherwise.

The belief in the association rule $A \rightarrow B$ determines the reliability of the transaction *T*. For the rule that satisfies the above conditions, the corresponding transaction is believed to be secure; if at least one condition is unsatisfied, the transaction will be treated as being insecure. The given *minsupp* and *minconf*

can be regulated to achieve different levels of security. In general, the larger their values are, the higher the requirement of security will be. By using association rule mining, the trust in the transaction can be transferred to compute the support and confidence of corresponding association rules. Therefore, it provides a novel and efficient way to analyse security protocols.

6.5 Algorithms and Experiments

6.5.1 Algorithms

Identifying frequent itemsets is one of the key issues in discovering association rules. There have been a number of algorithms developed for mining frequent itemsets in databases. Among them, *Apriori* is a widely-used algorithm for extracting frequent itemsets. However, the secure messages are different from traditional data due to the security properties. Hence this algorithm is inappropriate to deal with inconsistent secure messages. In this article, we extend the Frequent Patterns (FP) tree algorithm [9] to identify frequent itemsets from secure messages based, in which the properties of secure messages are taken into account.

Firstly, we need to generate all frequent items, which are supported by the transaction databases of principals. The *missing message*, *tampered message* and *private/public key* pairs mentioned above are considered to calculate the support and confidence of itemsets correctly. We assume that the secure messages are fresh and generated and sent by the sender and received and seen by whom it claims to be, and the keys have been authenticated to be valid and issued by the correct authorities.

Algorithm 5.1 *Mining Inconsistent Secure Messages*

begin

Input: D : *data set*; *minsupp*: *minimum support*; *minconf*: *minimum confidence*;

Output: *frequent itemsets*;

//Select the candidate transaction database.

let $D_c \leftarrow$ candidate transaction database with private key;

//Convert inconsistent secure messages.

//$1 \leq \mathrm{i} \leq n$, $1 \leq j \leq m$, in which n and m denotes the number and cardinality of databases respectively.

forall $D_i \in D - D_c$ **do**

forall $I_{ij} \in D_i$ **do**

if item $I_{ij} \neq null$ **then**

if I_{ij} is a *key* **then**

{

if $I_{ij} = I_{cj}$ or matches with I_{cj} in D_c then

$I_{ij} = K(D_i)$;
$I_{cj} = K(D_i)$;
}
end
end
//Construct FP-tree.
forall $I_{ij} \in D$ *do*
$F \leftarrow \{\}$;
$F \leftarrow F \cup$ frequent items of D_i;
sort items in F according to the frequency;
add items to FP-tree;
end
//Mining frequent patterns from FP-tree.
forall $node_i \in$ *FP-tree* **do**
process one node each time from bottom to the root;
output frequent itemsets;
end
end

This algorithm is used to extract frequent itemstes from transaction databases of principals. Before extracting frequent itemsets, the public/private key pairs in the databases need to be converted into a common assumed key $K(X)$ if it equals to the private key of candidate database or matches it. FP-tree algorithm [9] is used to mine frequent itemsets. It consists of two steps: (1) constructing FP-tree; and (2) mining frequent itemsets from FP-tree. The output only contains frequent itemsets so it is more efficient than the *Apriori* algorithm.

Preprocessing of secure messages needs $O(nm)$ time. As mentioned in [9], the search time of inserting a transaction *Trans* into the FP-tree is $O(|freq(Trans)|)$, where $freq(Trans)$ is the set of frequent items in *Trans*. The worst case is $|freq(Trans)|$ equals m. Thus, FP-tree construction needs $O(nm)$ time. The mining of frequent patterns phase is $O(m)$ time. Hence, our algorithm has the worst-case $O(nm + m)$.

6.5.2 Experiments

In this section, we study the efficiency and performance of our algorithm presented in Section 6.5.1 by verifying our algorithm against a stimulation dataset. The dataset is derived from a merchant's payment authorization process in SET protocol. The merchant authorizes the transaction during the processing of an order from a cardholder. *Third parties* here include the financial institutions and the processor of the transactions. SET aims at providing

confidentiality of information, ensuring payment integrity, and authenticating both merchants and payment gateway. For simplicity, the processes of *authorization request* and *authorization response* are considered only.

At the beginning, the merchant needs to generate an authorization request and send it to a payment gateway. When the payment gateway receives the authorization request, it verifies transaction identifier, and forwards an authorization request to the issuer through a payment system. The transactions details are briefly described as:

- *Authorization Request.* In order to authorize a transaction, the merchant generates an authorization request *AuthReq*, which includes the amount to be authorized, the transaction identifier from the *OI*. It is then combined with the transaction identifiers *TransIDs* and the hashing of the order information *OI*. *M* signs *AuthReq* and encrypts it with a randomly generated symmetric key k_2. This key is then encrypted with the gateway's public key $K_p(P)$. Finally, the merchant transmits the authorization request to the payment gateway *P*.
- *Processing Authorization Request.* The gateway decrypts the symmetric key k_2, and then decrypts authorization request using k_2. It uses the merchant public signature key $K_p(M)$ to verify the merchant digital signature. The gateway also verifies the merchant signature certificate and cardholder signature certificate to ensure that they have not expired. Then the gateway decrypts k_1 and cardholder account information with gateway private key $K^{-1}(P)$, and decrypts the the payment instructions *PI* (created by cardholder) using k_1. The gateway also verifies the transaction identifier received from the merchant by comparing it with the identifier in cardholder payment request.

The extracted secure messages from the above transaction databases include *OI*, *PI*, *AuthReq*, k_2, $K^{-1}(P)$, $K_p(P)$, $K_p(M)$, $K^{-1}(M)$ and k_1. They are stored in the following databases.

- $D_M = \{OI\ PI,\ AuthReq,\ K^{-1}(M),\ K_p(P),\ k_2\}$
- $D_P = \{OI,\ PI,\ AuthReq,\ K_p(M),\ K^{-1}(P),\ k_2\}$
- $D_T = \{OI,\ PI,\ AuthReq,\ K_p(M),\ K_p(P),\ k_2\}$

where key k_1 is not included in the above transaction databases since it is encrypted by $K_p(P)$ and is unknown to the merchant. Initially, each item has a value corresponding to the record in databases. On the other hand, the databases contain some inconsistent items such as missing items, tampered items and unmatched private/public key pairs. Before mining frequent itemsets, we need to convert the original item into unified data. Table 6.6 presents the items of transaction databases.

Suppose that the minimum support $minsupp = 50\%$ and minimum confidence $minconf = 60\%$. Let *i-itemset* be the frequent itemset that contains

Table 6.6. Payment Authorization in SET.

	OI	PI	$K(M)$	$K(P)$	k_2	$AuthReq$
D_M	OI	$\neg PI$	$K^{-1}(M)$	$K_p(P)$	k_2	$AuthReq$
D_P	OI	PI	$K_p(M)$	$K^{-1}(P)$	k_2	$\neg AuthReq$
D_{T_1}	OI	PI	$K_p(M)$	$K_p(P)$	k_2	$AuthReq$
D_{T_2}	$\neg OI$	PI	$K_p(M)$	$K_p(P)$	$\neg k_2$	$AuthReq$

k items. Hence we can get *frequent items* = $\{OI^3, PI^3, K(M)^4, K(P)^4, K_2^3, AuthReq^3\}$, in which the superscript represents the frequency of items. For example, $supp(OI\ ) = (1 + 1 + 1)/4 = 75\% > minsupp$ and $supp(PI) = (1 + 1 + 1)/4 = 75\% > minsupp$.

Based on the derived *frequent items*, we can find out all *frequent k-itemsets*($k \geq 2$) using the algorithm described in Section 6.5.1, in which no candidate itemset needs to be generated. The second step is to extract all association rules from the derived frequent itemsets. Several obtained association rules are listed below. The rules can be used to measure the trust in the goal that can be put on the protocol. On the other hand, some rules can be used to detect correlations between secure messages, which possibly imply potential security flaws. For simplicity, we present the findings of association rules in relation to *5-itemsets* only.

(1) $supp(OI \cup PI \cup K(M) \cup K(P) \cup k_2) = 50\% \geq minsupp$
(2) $supp(OI \cup PI \cup K(M) \cup K(P) \cup AuthReq) = 25\% < minsupp$
(3) $supp(OI \cup PI \cup K(M) \cup k_2 \cup AuthReq) = 25\% < minsupp$
(4) $supp(OI \cup PI \cup K(P) \cup k_2 \cup AuthReq) = 25\% < minsupp$
(5) $supp(OI \cup K(M) \cup K(P) \cup k_2 \cup AuthReq) = 50\% \geq minsupp$
(6) $supp(PI \cup K(M) \cup K(P) \cup k_2 \cup AuthReq) = 25\% < minsupp$

From the observation, only (1) and (5) are frequent itemsets, from which association rules can be derived. For example, $supp(\{OI, K(M), K(P), k_2\}) = 3/4 = 75\% > minsupp$, and $conf(\{OI, K(M), K(P), k_2\} \rightarrow AuthReq) = 67\% > minconf$. Therefore, $\{OI, K(M), K(P), k_2\} \rightarrow AuthReq$ can be extracted as a valid rule of interest. This rule indicates that $AuthReq$ is believed to be secure because OI, $K(M)$, $K(P)$, and k_2 are reliable. This rule in fact corresponds to a transaction in the payment authorization.

On the other hand, some discovered rules can imply potential correlation between secure messages. For example, $supp(\{OI, PI\}) = 0.5 > minsupp$, $supp(K(M), K(P), k_2) = 0.75 > minsupp$, and $conf(\{K(M), K(P), k_2\} \rightarrow \{OI, PI\}) = 0.67 > minconf$. Therefore, $\{K(M), K(P), k_2\} \rightarrow \{OI, PI\}$ is a valid rule of interest. This rule represents the correlation between $\{K(M), K(P), k_2\}$ and $\{OI, PI\}$. In other words, the trust in $\{K(M), K(P), k_2\}$ has an effect on the trust in $\{OI, PI\}$. Therefore, if PI and OI are found

to be highly inconsistent in transaction databases, $K(M)$, $K(P)$ and k_2 are suspected of being attacked.

Looking at (2), (3), (4) and (6), they are not frequent itemsets. Hence, they are ignored from discovering association rules. In other words, the secure messages in each of them take a low risk of being attacked.

6.6 Summary

The rapid growth of e-commerce brings out tremendous data exchanged between principals. Some data is secrets and must be protected from the malicious attacks. The formal analysis for security protocols is a useful way to find subtle flaws in the initial stage of protocol design. However, they usually make ideal assumption that the principals are honest and the data communication is secure. In practical trading environments, it may be impossible to exclude the possibility of missing and inconsistent data during transactions. The inconsistent secure messages have been a big challenge to the trust in electronic transactions. In that case, the traditional formal analysis may be unfit for dealing with the analysis with inconsistent messages. Thus, it requires us to develop new methods to enhance the current protocol analysis.

On the other hand, traditional formal approaches to analyse security protocols require users to predetermine authentication goals. However, it may be very difficult for the users to enumerate all suspectable points. In particular, the applications to which the security protocols can be put become varied and more complex, such as financial transactions. Fortunately, the transaction data of principals provide a valuable data set for further data mining. The trust in the goal of the protocol can be transferred to identify frequent patterns from the delivered secure messages between principals.

This chapter thus proposes a novel method to analyse security protocols in terms of association rule mining. The properties of secure messages are considered in the framework to examine secure messages before starting data mining. In particular, it takes into account the missing items and inconsistent items of secure messages and public/private key pairs. The discovered association rules are able to not only measure the trust in the corresponding transactions but also unveil potential associations between secure messages. The presented algorithm and experiments demonstrate that our methods can enhance the existing protocol analysis in an intuitive and systematic way.

7

Detection Models of Collusion Attacks

Security protocols have been widely used to safeguard secure electronic transactions. We usually assume that principals are credible and will not maliciously disclose their individual secrets to someone else. Nevertheless, it is impractical to completely ignore the possibility that some principals may collude in private to achieve a fraudulent or illegal purpose.

Collusion attack has been recognized as a key issue in e-commerce systems and increasingly attracted people's attention for quite some time in the literature on information security. Regardless of the wide application of security protocols, this attack has been largely ignored in the protocol analysis. There is a lack of efficient and intuitive approaches to identify this attack since it is usually hidden and too complicated to find. Therefore, it is critical to address the possibility of collusion attacks in order to analyse security protocols correctly. This chapter presents two frameworks by which to detect collusion attacks in security protocols. The possibility of security threats from insiders is especially taken into account. The results demonstrate that the frameworks are useful and promising in discovering and preventing collusion attacks, and enhancing the protocol analysis.

This chapter is organized as follows. Section 7.1 spells out our motivation to detect the collusion attack. Section 7.2 gives an overview to related work. In Section 7.3, we present the detection of collusion attack using data mining. The detection of collusion attack is converted into identifying frequent itemsets and matching rules. Section 7.4 presents a model by which to measure the probability of the attack using Bayesian networks. It helps users find the direct and indirect dependencies between secure messages. Section 7.5 concludes this chapter.

7.1 Introduction

With its rapid growth, electronic commerce (e-commerce) has come to play a central role in the global economy. However, the vast growth potential of

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 193–216, 2008.

electronic commerce is weakened due to security concerns, and security has become a high profile problem in e-commerce systems. For example, a customer's transaction record can be maliciously intercepted and revealed by computer or network hackers.

Threats to the security of electronic transactions can be classified into *internal* and *external*. The internal threat is clearly a danger, but most companies are more concerned about the external threat. Many companies feel reasonably safe from the internal threat, confident that it can be controlled through corporate policies and internal access control. Hence, they focus on the unknown outside users who may gain unauthorized access to the corporation's sensitive assets. Although it is usually very hard for a single principal to break through the protective barriers surrounding secure messages, a certain number of dishonest principals may put their respective secrets together to launch a collusion attack.

As a fundamental measure to fulfil corporate security objectives, security protocols have been commonly treated as a requisite of e-commerce systems. However, their designs create a difficult and error-prone task, and some subtle flaws have been found in a number of security protocols that were previously believed to be secure [22]. Subsequently, there has been considerable research on the analysis of security protocols by developing methodologies, theories, logics, and other supporting tools [22, 40]. These efforts are effective in overcoming weaknesses and reducing redundancies at the design stage of protocols. Among such efforts, the formal methods including theorem proving [22] and model checking [68] have been regarded as two of the most efficient approaches for protocol analysis. However, the possibility of internal threats, as mentioned above, has been greatly underestimated and ignored in the traditional approaches. They unrealistically assume that no principal can access secrets that exceed his/her usual legal authority. However, a user who attempts to obtain unauthorized data by colluding with other principals might discover more secrets that would otherwise remain protected. For example, principals A, B and C in Figure 7.1 can collude with each other to generate a message $\{m_1, m_2, m_3\}$ even though none of them previously knew this message individually. Therefore, detecting collusion attacks is critical so that reliable analysis of security protocols can be achieved.

There have been considerable efforts to ensure that digital data is secure from attack by collusion. A general fingerprinting solution used to detect any unauthorized copy is presented in [11]. A novel collusion-resilience mechanism using pre-warping was proposed to trace an illegal un-watermarked copy [25]. However, no work has been conducted to detect collusion attacks in security protocols.

The possibility that a collusion attack may occur is, in fact, determined by the degree of message sharing. Therefore, we can measure it by identifying frequent itemsets from transaction data sets [163]. The frequent itemsets

obtained can be used to search for collusion attacks. Also, we may need more illuminating ways to measure the likelihood that it may happen. In other words, we want to have a numeric estimation that the collusion attack happens, and to correctly capture the threat. Bayesian networks that have been widely used to represent the probabilistic relationships among a number of variables and conduct probabilistic inference with them [11] are eligible to fulfil this task. On the other hand, the available transaction databases of principals provide valuable but low cost data to perform Bayesian inference.

This chapter presents two frameworks to detect collusion attacks by using data mining [34] and Bayesian inference [35], respectively. As to the former, frequent itemsets that may launch attacks are extracted from the transaction data sets of principals. This reduces the search space. In particular, they are converted into the form of Prolog in order to match the rules in the established knowledge base. The case study demonstrates that our approach can complement the traditional analysis of security protocols. In the latter, it uses Bayesian networks to identify the hidden probabilistic dependency between secure messages and determine the likelihood that collusion attacks may occur.

7.2 Related Work

Digital watermark/fingerprint attack. Digital watermarking schemes allow tracing of illegally redistributed multimedia contents such as audio, image and video, and achieve copyright protection. To facilitate tracing of copyright violators, a number of watermarks carrying information about the transaction or content receipt can be embedded into the content before distribution. This form of personalised watermark is called fingerprint.

The collusion attacks allow a set of dishonest colluders to generate a new copy whose mark does not identify any of the colluders. This copy is to be redistributed illegally. To prevent collusion attacks, several fingerprinting codes to resist collusion attacks have been described [52, 105]. The identification of honest customers should be kept secret unless they act dishonestly. An honest customer should not be accused falsely by a dishonest vendor.

In traditional *symmetric fingerprinting*, the mark is embedded into the content by the merchant who later sell the market copy to the customer. The main problem of such schemes is that a dishonest merchant/custermer can redistribute a copy recently sold and can claim that it was customer/merchant who sent the copy. To prevent this situation, only the customer must know his/her marked copy. Such schemes are called *asymmetric schemes*.

A public fingerprinting infrastructure is proposed in [105]. It shows how to implement in practice asymmetric fingerprinting by using symmetric algorithms and trusted third parties. The customer and merchant must complete

registration with the registration authority (RA) and the fingerprinting authority (FA), respectively. Then, the merchant can use FA to mark digital content and the customer can obtain an unchangeable pseudonym PS_c from RA. The protocols below will be used when a customer decide to make a purchase.

- **Mark embedding**. To make a purchase, a customer must generate a message including a description of the product X, perform a payment. The merchant must verify the validity of the signed messages and the FA must construct a collusion-secure fingerprinting code. Finally, a marked copy $\bar{X}$ is sent to the customer.
- **Identification of dishonest customers**. If the merchant finds an illegally redistributed copy of the multimedia content, this protocol is used to identify dishonest customers. It aims to identify either the owner of $\bar{X}$, or some of the colluders that contributed to create an altered copy of $\bar{X}$, $\hat{X}$. The primary procedures include:
  1. M sends the illegal copy to FA.
  2. FA recovers the embedded mark m from the illegal copy. A robust watermarking scheme should be able to indicate whether a mark is embedded in the object or not.
  3. If a mark has been found, the message is decoded so that an illegal codeword is obtained. The mark is decoded as it may have been altered as a result of a collusion attack.
  4. The pseudonym related to the guilty codeword is sent to RA to disclose the identity of guilty customer.

Collusion attack and performance evaluation. In a collusion attack, different fingerprinted copies of same host data are jointly processed to remove the fingerprints or hinder their detection. Several existing collusion attack techniques are listed below.

- *Averaging collusion attack*. The attacked image $\hat{E}$ is created by averaging K fingerprinted images. Under this attack, each colluding fingerprint has a contribution of strength $1/K$ in the attacked sequence $\hat{W}$. This attack requires a large number of fingerprinted images to reduce the correlation coefficient substantially.
- *Maximum-minimum collusion attack*. The attacked image is created by talking the average of the maximum and minimum values across the component values of the fingerprinted image. The attack considers the possibility that the fingerprinted values of a particular position across the fingerprinted images may not be evenly distributed around its mean value. Therefore, this attack uses the middle point between the maximum and minimum rather than using the mean value.

- *Negative-correlation and Zero-correlation collusion attack.* This attack is to drive the correlation coefficient to negative value using as few as five fingerprinted images [146]. The attacked images take on the least likely values across the fingerprinted copies as the attacked values.

In [156], a new collusion attack scheme is proposed and evaluated in terms of fingerprint detectability and imperceptibility after attack is assessed.

- *Detectability.* An attack is considered effective if the original fingerprint cannot be detected from the attacked data, whereas this may not necessarily mean that the original fingerprint is completely removed.
- *Imperceptibility.* An attack is successful if the perceptual quality of the attacked data does not deteriorate substantially from the original fingerprinted data. For example, the visual quality of the attacked image must not drop much in contrast to the fingerprinted image.

A correlation coefficient and a weighted PSNR (Peak signal-to-noise ratio) are used to measure the fingerprint detectability and visual quality, respectively. The expression for weighted PSNR (wPSNR) is computed by using the formulae below.

$$wPSNR = 20log_{10}\frac{max(p)}{\|NVF(p' - p)\|}$$

$$NVF_{j_1,j_2} = \frac{1}{1 + \sigma^2_{L_{j_1,j_2}}}$$

where p represents the original image pixels, p' denotes pixels of the tested image, $\| \ \|$ denotes the root-mean-square value. In the expression of NVF (noise visibility function), $\sigma^2_{L_{j_1,j_2}}$ represents the local variance of an image in a window centered on the pixel with coordinates (j_1, j_2).

Let W_T be a colluding fingerprint. The correlation coefficient of $\hat{W}$ and W_T can be calculated by using the formula below.

$$C(\hat{W}, W_T) = \frac{cov(\hat{W}, W_T)}{\sqrt{var(\hat{W}) * var(W_T)}}$$

where $cov(\hat{W},\ W_T)$, $var(\hat{W}$ and $var(\ W_T)$ represent the covariance of $\hat{W}$ and W_T, the variance of $\hat{W}$ and the variance of W_T, respectively.

The performance evaluation of collusion attacks depends on the number of colluding parties to achieve near-zero-zero detection and the resultant visual quality of the attacked data.

It is observed that most of existing studies in collusion attacks mainly focus on either developing new fingerprinting methods for copyright protection or proposing schemes for evaluating the strength of the fingerprint against

different collusion attacks. Nevertheless, current approaches still have some limitations.

- No matter symmetric fingerprinting or asymmetric fingerprinting, we are still concerned about their robustness to some extent due to varied collusion attacks.
- Existing evaluation schemes can assist in comparing the strength of different fingerprinting techniques against specific collusion attacks, whereas it is unrealistic for a fingerprint to be robust in all cases.

Unlike the multimedia content, transaction data is often shared by more than one principal. The collusion attacks on the data are usually hidden to us and have no obvious symptoms like visual quality of fingerprints. Unfortunately, no much work has been found to identify collusion attacks in an intuitive way. As a result, it is necessary to identify potential collusion attacks by analysing the available transaction data.

7.3 Identification of Frequent Patterns for Collusion Attack Detection

7.3.1 Basic Concepts

The processing of messages usually includes generation, sending, receiving and authentication in e-commerce systems. They are transmitted via either plaintext or ciphertext. The following rules are derived from the ENDL logic [29] in Chapter 3 and present the fundamental operation of the messages.

(1) *Generation Rule*. If message m is generated by X, X must know m.

$$\frac{generate(X,m)}{know(X,m)}$$

(2) *Delivery Rule*. If X knows message m and sends m to receiver Y, Y knows the message m.

$$\frac{know(X,m) \wedge send(X,Y,m)}{know(Y,m)}$$

(3) *Public Key Rule*. If Y knows message m, Y can sign this message using his/her private key.

$$\frac{know(Y,m)}{know(Y,S(m,K^{-1}(Y)))}$$

(4) *Encryption Rule*. If Y knows message m and a key k, Y can encrypt this message using k.

$$\frac{know(Y,m) \wedge know(Y,k)}{know(Y,E(m,K))}$$

(5) *Belief Rule.* X generates message m and sends it to Y. If Y sees this message and m is fresh, Y believes X in the message m.

$$\frac{send(X,Y,m) \wedge know(Y,m) \wedge fresh(m)}{believe(Y,X,m)}$$

This rule indicates that principal Y believes that the message m from X is not a replay. The timestamp is usually used to ensure the freshness of secure messages [40].

(6) *Certificate Rule1.* If CA_2's certificate is signed with CA_1's private key, and Y verifies CA_2's public key using CA_1's public key, Y believes CA_2's public key.

$$\frac{signs_c(CA_1,CA_2,Cert(CA_2)_{CA_1}) \wedge verify(Y,CA_2,K_p(CA_2))}{believe(Y,CA_2,K_p(CA_2))}$$

(7) *Certificate Rule2.* If X's certificate is signed with CA_2's private key, and Y verifies X's public key, principal Y believes X's public key.

$$\frac{signs_p(CA,X,Cert(X)_{CA}) \wedge verify(Y,X,K_p(X))}{believe(Y,X,K_p(X))}$$

The first four rules describe the generation, transition and basic encryption operation of messages. The remainder validate the belief surrounding the message freshness and the validity of the principal's public keys. More rules can be found in Chapter 4, thus they are not described here due to limited space.

A collusion attack usually consists of an intruder Z, a group of principals $P = \{P_1, \cdots, P_n\}$, and a threshold of collusion attack k, $1 \leq k \leq n$. Figure 7.1 presents an instance of how A, B and C can collaborate to disclose a secret to Z. Note, not all combinations of k principals are able to break the secret. In a simple way, a collusion attack may include the following factors:

- Principal P: the user who participates in the transaction using electronic transaction protocols.
- Intruder Z: a user who intends to collect messages from a certain number of dishonest principals to launch the attack.
- Threshold k: the minimum number of principals who can collude to perform a collusion attack.

Definition 7.1. *The access structure Γ of the group $P = \{P_1, \cdots, P_n\}$ denotes principals who may jointly recover secret s by putting their individual secrets together. M_X denotes a set of secure messages of X. $A(k, n)$ is the threshold scheme that allows the secret to be recovered if the currently active subgroup $A \subset P$ has $k < n$ principals.*

$$\Gamma = \{A \mid \alpha_1 \wedge \cdots \wedge \alpha_k \rightarrow s, \alpha_i \subset M_{X_i}, X_i \in A, 1 \leq k \leq n\}$$

where α_i represents a subset of messages from M_{X_i}.

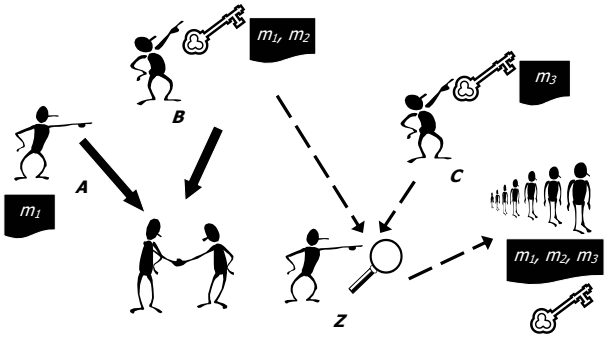

Fig. 7.1. A collusion attack handled by A, B and C

Example 7.1. Let $M_{Tom} = \{Jim, order\}$, $M_{Bob} = \{order, one, textbook\}$ and $M_{Alice} = \{biology, textbook\}$ be secure messages. Nobody can obtain a full understanding of this order alone but '*Jim orders one biology textbook*' can be derived by integrating $\{Jim\} \subset M_{Tom}$, $\{order, one\} \subset M_{Bob}$ and $\{biology, textbooks\} \subset M_{Alice}$ together. Hence, we have $\Gamma = \{Tom, Bob, Alice\}$ and $k = 3$.

From the observation, the above instance cannot generate a collusion attack since it requires all principals to participate in generating the order. *Tom* and *Bob* for instance may cooperate to get '*Jim orders one textbook*' but never know it is a textbook of biology. According to *Definition 8.1*, a collusion attack must satisfy three conditions.

(1) $\alpha_1 \wedge \cdots \wedge \alpha_k \rightarrow s$, $\alpha_i \subset M_{X_i}$, $X_i \in A$;
(2) $1 \leq k \leq n$; and
(3) $\forall\ \alpha_i$, α_i must belong to more than one principal at least.

If α_i belongs to a single participant X_i only, it is not difficult to confirm that X_i participated in the attacks. Nevertheless, some secrets may be shared among several principals. In this case, it is hard to determine who should be responsible for the disclosure of the secrets. This may endanger transaction security, as the intruder Z is able to generate the secret s without going through the usual authentication process.

7.3.2 A Framework to Detect Collusion Attacks

As mentioned above, each α_i used to launch the attack must belong to more than one principal. Hence, it is necessary to discover the α_i that is shared by more than one principal. The secure messages from each principal can be viewed as a transaction database. Therefore, the detection of α_i can be converted to identify frequent k-itemsets ($2 \leq k \leq n$). The detection of collusion attacks may include the following phases:

1. identify frequent k-itemsets from the transaction database of principals;
2. construct knowledge based on the given inference rules, and;
3. detect collusion attacks by matching frequent itemsets with the knowledge base.

Identifying Frequent Itemsets. Let $I = \{i_1, \cdots, i_n\}$ be a set of items and D be a collection of transactions, called the transaction database. Each transaction $T \in D$ consists of a collection of items. Let $A \subseteq I$ be an itemset. We can say that a transaction T contains A in the case of $A \subseteq T$. An itemset A in a transaction database D has a support, denoted as $supp(A)$. Hence, we have:

$$supp(A) = |T_A|/|D|\% \tag{7.1}$$

where T_A represents transactions in D, which contain itemset A.

An itemset A in D is called a frequent itemset if its support is equal to, or greater than, the minimum support *minsupp* that is designated by a user or experts. The details can be found in the support-confidence framework [3, 163]. In this chapter, Frequent Patterns (FP) tree algorithm [67] is used to identify frequent itemsets from transaction data. Nevertheless, we do not focus on discussing how to select an optimal algorithm to discover the rules of interest as in traditional data mining.

From the observation, *minsupp* needs to be specified so that frequent itemsets can be identified. According to the prerequisites of collusion attacks mentioned above, each message subset α_i must belong to at least two principals. Suppose there are $n(n \geq 3)$ principals in a transaction. Then,

$$minsupp = \frac{m}{n} \tag{7.2}$$

Here, n must be equal to, or greater than, 3 for it is impractical for a collusion attack to take place in a transaction that includes only two principals. In other words, it is not difficult to detect this attack if it really happens. On the other hand, the value of $m \geq 2$ can be tuned by users in terms of different level of security requirements. The bigger its value is, the more the identified frequent itemsets will be.

Example 7.2. Suppose there are four principals, P_1, P_2, P_3, and P_4, in a transaction T. Their datasets are represented by $\{\alpha, \beta, \gamma, \mu\}$, $\{\alpha, \beta, \mu\}$, $\{\beta, \gamma, \nu\}$ and $\{\alpha, \gamma\}$, respectively. Let $m = 2$. According to formula (8.2), we have $minsupp = 2/4 = 0.5$. Then, $supp(\alpha) = 3/4 = 0.75 > minsupp$, $supp(\beta) = 3/4 = 0.75 > minsupp$, $supp(\gamma) = 3/4 = 0.75 > minsupp$, $supp(\mu) = 2/4 = 0.5 \geq minsupp$ and $supp(\nu) = 1/4 = 0.25 < minsupp$. Thus, frequent 1-itemsets include $\{\alpha\}$, β, γ and μ. In the same way, we can identify frequent 2-itemsets, such as $\text{supp}(\alpha \cup \beta) = 2/4 = 0.5 \geq minsupp$.

7.3.3 Dealing with Knowledge and Facts

This section suggests how to construct a knowledge base and manipulate derived facts from transaction databases. For brevity, it assumes that communication channels and keys are secure and reliable. Additionally, the secure messages are assumed to be fresh, and there are correct associations between public keys and principals. Consequently, the belief in message freshness and the validity of a principal's public keys are not discussed here. The details can be seen from Chapter 3.

As mentioned in Chapter 4, a *knowledge base* comprises the knowledge that is specific to the domain of application, including such things as *facts* in the domain, and *rules* that describe the relations or phenomena in the domain. The inference rules of *knowledge base* consist of the basic manipulation of secure messages in security protocols. *Facts* are defined as general knowledge that is commonly accepted by people. For example, '*Alice* knows her own public/private keys'. Suppose R denotes the inference rules of a knowledge base. Then,

$$R = \{rule_1, rule_2, \cdots, rule_n\}$$

where the rules of a knowledge base are of the form:

$$rule_i = \{(N, [Condition_{ij}], Conclusion_i) | \ 1 \leq i \ n, 1 \leq j\}$$

where $Condition_{ij}$ is a set of simple assertions linked by logic connectives, $Conclusion_i$ is a simple assertion without logic connectives, and N is the rule name. The assertions in rules can be terms that contain variables.

Example 7.3. The generation rule and delivery rule in Section 7.3.1 can be written as $rule1 = (1, [generate(X, m)], know(X, m))$ and $rule2 = (2, [know(X, m), send(X, Y, m)], know(Y, m))$ respectively.

Each *transaction database* comprises a collection of secure messages from a corresponding principal. As mentioned above, we aim to identify frequent itemsets from transaction databases. The detection of collusion attacks is implemented by matching derived frequent itemsets with knowledge bases. Suppose the transaction database T contains m principals. Then,

$$T = \{M_{P_1}, \cdots, M_{P_m}\}$$

where each M_{P_i} $(1 \leq i \leq m)$ denotes the set of secure messages from principal P_i.

Example 7.4. Based on Example 7.2, we have $T = \{M_{Tom}, M_{Bob}, M_{Alice}\} = \{\{Jim, order\}, \{order, one, textbook\}, \{biology, textbook\}\}$.

Detecting Collusion Attacks. The established knowledge base and derived frequent itemsets will be used to detect potential collusion attacks in security

protocols. In this chapter, the intrinsic inference mechanisms of Prolog [20] are used to manipulate the knowledge base and frequent itemsets. Nevertheless, the frequent itemsets need to be converted to the forms of predicate that conform to Prolog. In addition, the host names of secure messages are required to identify principals who may involve themselves in the attack.

Definition 7.2. *Suppose* $F_k = \{\{\alpha_1, \cdots, \alpha_k\} \mid supp(\alpha_1 \cup \cdots \cup \alpha_k) \geq \text{min-supp}, \alpha_i \in T, 1 \leq i \leq k\}$ *denotes a set of frequent k-itemsets from transaction T. Let* $P = \{P_1, \cdots, P_n\}$ *be a group of principals who participate in this transaction. Then,*

$$know(P_j, \{\alpha_1, \cdots, \alpha_k\}) \; iff \; \{\alpha_1, \cdots, \alpha_k\} \; is \; a \; frequent \; itemset \; of \; P_j \quad (7.3)$$

In this definition, the predicate $know(P_j, \{\alpha_1, \cdots, \alpha_k\})$ denotes that the principal P_j knows the message $\{\alpha_1, \cdots, \alpha_k\}$. According to the formula (7.2), $\{\alpha_1, \cdots, \alpha_k\}$ ought to be known by more than one principal.

Example 7.5. As mentioned in Example 7.2, α, β, γ, and μ are frequent 1-itemsets and $\alpha \cup \beta$ is frequent 2-itemsets. Hence, we have $know(P_1, \alpha)$, $know(P_1, \beta)$, $know(P_1, \gamma)$, $know(P_1, \mu)$, $know(P_2, \alpha)$, $know(P_2, \beta)$, $know(P_1, \alpha \cup \beta)$ and $know(P_2, \alpha \cup \beta)$ after conversion.

The converted frequent itemsets can be collected via interaction using a user interface. Additionally, the fact database is emptied before collection. Once users submit a detection request, we need a reasoning procedure to manipulate the knowledge base and derived frequent itemsets efficiently. As for if-then rules, there are two basic ways of reasoning [20], including backward chaining, and forward chaining.

In contrast to forward chaining, backward chaining starts with a hypothesis and works backwards, according to the rules in the knowledge base, toward easily confirmed findings. Thus, the backward chaining is chosen as the reasoning method in our detection model, which searches for the goal we want to verify. The detection starts with a pre-defined suspect secure message that may suffer from collusion attacks. If it eventually reaches the goal, the authentication succeeds. In other words, a collusion attack is found. Otherwise, if the goal cannot be proven, based on existing information, it is natural to conclude that no collusion attack occurred in the current transactions.

7.3.4 A Case Study

To illustrate the application of the proposed approach, an instance in respect of an online transaction intersected from SET protocol [140] is presented. It is sufficiently flexible for us to analyse other security protocols due to the extensibility of knowledge base.

This example presents a registration form request handled by a cardholder C. It aims to obtain a valid registration form from certificate authority (CA) to complete registration. If the registration form can be obtained, it is not difficult to initiate a certificate request to gain valid certificates issued by CA.

As described in [140], the transited secure messages comprise primary account number (PAN), registration form request ($RegFormReq$), symmetric key k_1 and public key-exchange key of CA, namely $Kpb(CA)$). Only CA, C and the *Issuer* know PAN, which is effectively obfuscated using a blinding technique. Suppose there are four principals in this process. The set of secure messages from each principal can be regarded as a transaction database, such as $M_{P_1} = \{PAN, RegFormReq, k_1, Kpb(CA)\}$ and $M_{P_4} = \{null, null, null, Kpb(CA)\}$. Table 7.1 presents the secure messages in the transaction databases.

Table 7.1. Secure messages in registration form request.

Principal	*PAN*	*RegFormReq*	k_1	$Kpb(CA)$
P_1	*PAN*	*RegFormReq*	K_1	$Kpb(CA)$
P_2	*PAN*	*RegFormReq*	K_1	$Kpb(CA)$
P_3	*PAN*	*RegFormReq*	*null*	$Kpb(CA)$
P_4	*null*	*null*	*null*	$Kpb(CA)$

The goal is to identify frequent itemsets from Table 7.1. Let $m = 2$. According to the formula (7.2), $minsupp = 2/4 = 0.5$. As a result, the frequent itemsets can be derived using the Frequent Patterns (FP) algorithm [67].

- frequent 1-itemsets: $\{PAN\}$, $\{RegFormReq\}$, $\{k_1\}$, $\{Kpb(CA)\}$;
- frequent 2-itemsets: $\{PAN, RegFormReq\}$, $\{PAN, k_1\}$, $\{PAN, Kpb(CA)\}$, $\{RegFormReq, k_1\}$, $\{RegFormReq, Kpb(CA)\}$, $\{k_1, \{Kpb(CA)\}$;
- frequent 3-itemsets: $\{PAN, RegFormReq, k_1\}$, $\{PAN, RegFormReq, Kpb(CA)\}$, $\{RegFormReq, k_1, Kpb(CA)\}$;
- frequent 4-itemsets: $\{PAN, RegFormReq, k_1, Kpb(CA)\}$.

After obtaining these frequent itemsets, it is necessary to convert them to the forms of predicate as Prolog. $\{PAN\}$, for example, can be transformed to $know(P_1, PAN)$, $know(P_2, PAN)$, $know(P_3, PAN)$ and $know(P_4, PAN)$.

The knowledge base that consists of inference rules and facts can be constructed via a user interface mentioned in Section 7.3.2. Once the processes are completed, the user can submit a detection request:

$?- Detection(E(RegFormReq, k-1))$

Backward chaining search is applied here. The detection model attempts to find matched rules to the verified goal from the knowledge base. The detection system finally returns a '*true*' value for $Detection(E(RegFormReq, k_1))$ since {RegFormReq, k_1} is a frequent itemset and satisfies the encryption rule of knowledge base. Finally, an early warning of collusion attacks is sent to the user. In the same way, the user can lodge another request:

$$?- \ Detection(S(<k_1, PAN>, Kpb(CA)))$$

In this request, the detection system needs to deal with the matching of two frequent itemsets including $\{k_1, PAN\}$ and $\{Kpb(CA)\}$. Finally, it is ascertained that the transaction contains potential collusion attacks since both $\{k_1, PAN\}$ and $\{Kpb(CA)\}$ are frequent itemsets and satisfy the public key rule of the knowledge base. Certainly, the user can put in any detection requests, and more exercises are left to the reader.

7.4 Estimation of the Probability of Collusion Attacks

Collusion attack has been recognized as a key issue in e-commerce systems and has increasingly attracted attention for quite some time in the literature on information security. Although the framework developed in Section 7.3 provides a useful way to identify collusion attack, there is a lack of methods to measure the likelihood of the attacks. Thus, we address this critical issue by using a compact and intuitive Bayesian network-based scheme in this section. This assists in not only identifying the secure messages that may lead to the attack but also giving the degree of dependency to measure the occurrence of a collusion attack.

7.4.1 Motivations

A secure e-commerce system relies on the valid and flawless combination of security protocols, secure and unobstructed communication channels and trustworthy principals [47]. However, it is not easy to guarantee all these factors due to the increasingly complicated security protocols and hostile environment [1].

The application of formal analysis for security protocols starts with the analysis of key distribution protocols for communication between two principals [112]. For example, A and B want to communicate with each other in a secure manner. They can reach this goal using a session key, which can be obtained from a common key server or generated by themselves. Despite the seeming simplicity of this problem, it is in fact not easy to handle because, in a hostile environment, the intruders may intercept, alter, or delete messages. Thus, the previous studies of protocol analysis focus on this topic.

As mentioned in Section 7.3, networks must defend not only against intruders who may impersonate an honest principal or attempt to learn secrets (external threats), but they must be robust against a group of dishonest principals who may collude together to uncover secrets (internal threat). Although the latter is viewed as a danger, most companies feel reasonably confident that the internal threat can be controlled through the corporate policies and internal access control. Thus, they concentrate on the unknown outside user who may obtain unauthorized access to the corporation's sensitive assets. Although it is difficult for an individual to break the protection over secrets, it is feasible to deploy an attack in collusion with a certain number of dishonest principals. In this way, he/she can access the unexpected secrets that exceed his/her legal authority [46]. Unfortunately, this threat has been largely neglected.

Instead of keys, security association has become a potential way to detect threats. Section 7.3 mentions the efforts made to ensure the digital data is secure in the context of collusion. Unfortunately, these measures may be too expensive and difficult to use. To find collusion attacks, we need a numerical estimation of the occurrence of the attack, and the ability to capture the threat correctly .

The main idea in our approach is that a collusion attack usually arises from attacks on encryption keys and delivered messages. Suppose message m_1 is shared by principal P_1 and P_2, and message m_2 is shared by P_3 and P_4. If m_1 and m_2 are revealed to a hostile intruder, the attack may occur. Thus, the dependencies can be used to evaluate the occurrence of the attack. Bayesian networks that have been widely used to represent the probabilistic relationships among variables and conduct probabilistic inference with them are eligible to fulfil this role [119]. The transaction databases of principals provide valuable data to perform Bayesian inference.

We aim to use Bayesian networks to find the dependency model (probabilistic dependence) between secure messages. We measure the collusion attack by observing the decrease of probability in case of removing the corresponding arcs of the model. This assists in discovering the collusion threat and enhancing the protocol analysis.

7.4.2 Preliminaries

There are a number of security considerations to guarantee secure transactions. In general, they can be classified into the following three categories:

- the robustness of cryptographic algorithms;
- the security protocols dependent on the cryptographic algorithm; and
- the correct association of specific principals with specific cryptographic keys.

The strength of the cryptographic algorithm relies on the complexity of cryptanalysis with respect to mathematic computation. In this article, we assume the cryptographic algorithm in e-commerce systems is robust in the usual sense. Thus, we do not discuss the threats arising from the aspect of cryptographic algorithms.

Although the traditional formal methods for protocol analysis have been useful in finding subtle threats in the design stage of protocols, they are criticized due to their ideal assumption of honest principals and secure communication channels. However, in a hostile/uncertain environment, the message can be missing or tampered with, and the principals can become dishonest.

Another important security issue of e-commerce systems is the correct association of cryptographic keys with principals. This is usually achieved using certificates digitally signed by a chain of certification authorities. The validity of a principal's public key can be validated by checking the certificate through a hierarchy of trust described in Chapter 4. Each certificate is linked to the signature certificate of the entity that digitally signed it. By following the trust tree to a known trusted party, one can be assured that the certificate is valid.

Given an electronic transaction including n principals, a principal P_i shares a set of secure messages m_{ij}, $1 \leq i \neq \mathrm{j} \leq k$, with the principal P_j, in a probability p_{ij}. The probability p_{ij} denotes the degree to which a message is shared between principals. For example, suppose the message $\{m_1, m_2\}$ is shared between not only P_1 and P_2 but also P_1 and P_3. Thus, the probability that $\{m_1, m_2\}$ is revealed by P_1 and P_2 is 17% since it can be derived rom not only P_1 and P_2, but also P_1 and P_3. In particular, the pairwise keys are only shared between the principal and the corresponding authorities. Consequently, they have a low possibility of being divulged, in contrast to general secure messages such as the principals' identity.

Unlike general secure messages, the pairwise keys consist of a public key and a private key. The former can be open to the public but the latter can only be known by the legitimate principal. Suppose all the principals who know X's public key are called X's neighbours. Note that A can have one or more than one public key in different cases. For brevity, this article assumes that each principal has only one public key in a transaction. Figure 7.2 presents principal A's and B's neighbours. Let $N(X)$ and $U(X)$ be the set of neighbours of X and the set of usable pairwise key of X, respectively. Thus, we have $N(A) = \{C, D\}$ and $N(B) = \{E, F\}$. G is not a neighbour of A because he/she shares a symmetric key k rather than a pairwise key with A. If A and B share each other's secrets, then A and B can communicate with each other's neighbours. In the same way, A can communicate with E and F by pretending to be B, and B can communicate with C and D by pretending to be A. $U(A) = \{K_{CA}, K_{DA}\}$ and $U(B) = \{K_{EB}, K_{FB}\}$ prior to collusion, but $U(A) = U(B) =$

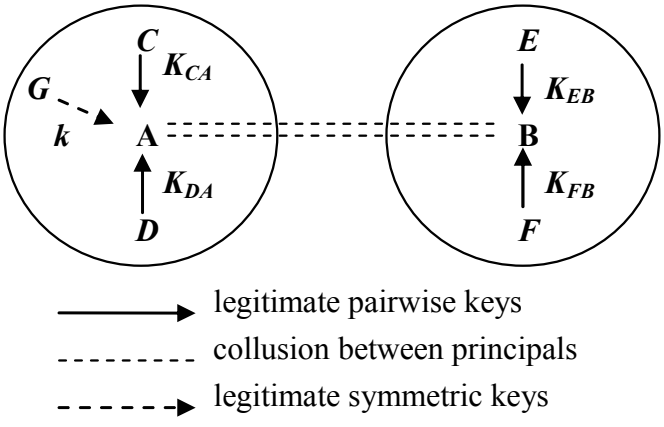

Fig. 7.2. An example of principal A's and B's neighbours

$\{K_{CA}, K_{DA}, K_{EB}, K_{EB}\}$ via collusion. Thus, a certain number of dishonest principals can obtain unexpected secrets via collusion.

In addition, there may exist potential correlations between secure messages. For example, an encrypted message has to be decrypted by the combination of the encrypted message and a corresponding pairwise key shared between the sender and receiver. It is observed that there should be dependencies between the messages shared by principals. Although it is not easy to confirm precisely the principals who can conduct the attack, it is feasible to find the probable messages causing the attack and evaluate its occurrence probability. A Bayesian network is appropriate to perform this role. With prior knowledge (the probability of divulging of secure messages) from the transaction databases, we can work out the subjective probability, namely the likelihood of collusion attack.

7.4.3 Identifying Collusion Attack Using Bayesian Network

Structure. A Bayesian network comprises a set of nodes and a set of directed links. Each node represents a variable from the domain and each link connects pairs of nodes, representing the direct dependencies between variables. Usually, the strength of the relationship between variables is quantified by conditional probability distribution associated with each node. Only the nodes that take discrete values are considered here. Note, we will consider only nodes that take discrete values in this article. Thus, a given node (secure message) represents a proposition, taking the binary values R (revealed by a certain number of principals) and NR (not revealed by the principals). For example, in an electronic transaction domain, a node called *symmetric* key might represent the proposition that the key is broken.

There are also direct relationships between principals and cryptographic keys as well as the dependencies between secure messages. To represent the described dependencies, we can construct a Bayesian network by creating a directed acyclic graph (DAG). There are three primary steps to build the network.

1. Identifying the variables of interest by answering the question of what the nodes are to represent and what values they can take.
2. Generating the network structure by deciding the parent nodes and child nodes.
3. Qualifying the relationships between connected nodes by calculating the conditional probabilities

Suppose a transaction T consists of a set of principals $P = \{P_1, \cdots, P_n\}$ and a collection of secure messages $m = \{m_1, \cdots, m_k\}$. Each principal P_i has a set of secure messages $M(P_i) \subseteq M$. For example, $\forall\ P_i \in P$ and $P_j \in P$, we have $M(P_i) = \{m_1, m_2, m_3\}$, $M(P_j) = \{m_1, m_2\}$ and $M(P_i) \cap M(P_j) = \{m_1, m_2\}$. It provides the intruder Z an opportunity to know the message m_1 or m_2 from P_i or P_j, whereas the user may not decide exactly who did it. Thus, we need to cope with uncertainty. In other words, we shall have to deal with incomplete evidence. As a result, there are two edges from $\{m_1, m_2\}$ to m_1 and m_2 in the network. The connections represent that m_1 and m_2 are the direct cause of $\{m_1, m_2\}$. From the observation, all messages included in the transaction are viewed as the nodes in the network and they take the binary values R and NR as mentioned above.

In addition, most messages are transmitted by cipher text to defend them against malicious attacks. Without exception, if an intruder knows a key and the encrypted message by the key, he/she can know the plain text of the message using decryption. For example, suppose $M(P_i) = \{K_{ij}, K_{il}, K_{ie}, E(m_3, K_{ij})\}$, $M(P_j) = \{K_{ij}, K_{jh}, K_{jf}, E(m_3, K_{ij})\}$ and $M(P_i) \cap M(P_j) = \{K_{ij}, E(m_3, K_{ij})\}$. In order to know m_3, the intruder must know K_{ij} and $E(m_3, K_{ij})$ or obtain m_3 from its generator directly. Thus, there are two edges from $\{K_{ij}, E(m_3, K_{ij})\}$ to m_3 and from the generator to m_3. It indicates that $\{K_{ij}, E(m_3, K_{ij})\}$ and the generator are the direct cause of m_3. Also, there is an edge from $\{m_3, K_{ij}\}$ to $E(m_3, K_{ij})$. The above dependencies can be extended to include the other principals and the secure messages included in the transaction T. In particular, if P_i and P_j conduct a collusion attack, the network can become complex because more dependencies will be included and they can communicate with neighbours of the other side. Finally, we can construct the whole DAG for the transaction T. This actually answers the second question. In other words, a node is a parent of a child, if there is an edge from the former to the latter.

So far, we have presented the identification of variables in the network and the generation of the network structure. The remaining work is how to

qualify the dependencies between linked nodes by calculating the conditional probabilities.

Ascertaining the Probability of Variables. To measure the collusion attack in an intuitive way, it is necessary to work out the probabilities for each variable in the obtained DAG. As described above, each variable can take binary values including R and NR, which are represented by the symbols + and -, respectively, in the following for the reason of simplicity. The probability corresponding to each node is a conditional probability, which relies on all its parent nodes in DAG that connect to this node. We need to look at all the possible combinations of values of those parent nodes (instantiation of the parent set). The example below is used to illustrate how to compute the conditional probabilities of variables in DAG.

For example, there are three principals P_1, P_2 and P_3 in a transaction T. The set of secure messages of principals are $M(P_1) = \{m_1, m_2, m_3, m_4\}$, $M(P_2) = \{m_1, m_2\}$, and $M(P_3)= \{m_1, m_5, m_6\}$. If an intruder wants to generate the message (m_1, m_2), he/she has to know both m_1 and m_2. Thus, we can generate a rule $m_1 \wedge m_2 \rightarrow \{m_1, m_2\}$, which represents the message $\{m_1, m_2\}$ arises from the combination of m_1 and m_2. On the other hand, each variable actually has binary values as mentioned above. The *divulgence* and *non-divulgence* of m_1 are denoted by m_1^+ and m_1^-, respectively. Consequently, the formula $P(m_1^+, m_2^-)$ represents the probability that m_1 is divulged but m_2 is not. Usually, if an intruder knows the combination of m_1 and m_2, namely $\{m_1, m_2\}$, he/she must know every element of this message. In the presence of a collusion attack, the intruder must have knowledge about all connected nodes in an alternative network path.

After specifying the topology of the BN, the next step is to qualify the dependencies between linked nodes. As we are only considering discrete variables at this stage, it has the form of a conditional probability table (CPT). For example, consider the $\{m_1, m_2\}$ node in Figure 7.1. Its parents are m_1 and m_2 having the possible joint values $\{<R, R>, <R, NR>, <NR, R>, <NR, NR>\}$. Table 7.1 specifies in order the probability of divulgence of $\{m_1, m_2\}$ for each of these cases to be $<0.2, 0.1, 0.1, 0.02>$. Thus, the probability of *no divulgence* of $\{m_1, m_2\}$ is given as one minus the above probabilities in each case, namely $<0.8, 0.9, 0.9, 0.98>$.

Table 7.2. A CPT of the node $\{m_1, m_2\}$ in Figure 7.1.

m_1	m_2	$P(\{m_1, m_2\} = R \mid m_1, m_2)$	$P(\{m_1, m_2\} = NR \mid m_1, m_2)$
R	R	0.2	0.8
R	NR	0.1	0.9
NR	R	0.1	0.9
NR	NR	0.02	0.98

Definition 7.3. *Let T be a transaction, $P_1, \cdots$ and P_n be principals in T and $M(P_1), \cdots$ and $M(P_n)$ be the set of messages of $P_1, \cdots$ and P_n, respectively. Suppose $m_1, \cdots$ and m_k are the cause (parent nodes) of the node $\{m_1, \cdots, m_k\}$, $m_i \in M(P_1) \cup \cdots \cup M(P_n)$. Thus, we have*

$$P(\{m_1, \cdots, m_k\}|m_1, \cdots, m_k) = \frac{P(\{m_1, \cdots, m_k\}) * P(m_1, \cdots, m_k|\{m_1, \cdots, m_k\})}{P(m_1, \cdots, m_k)}$$

From the observation, there are three probabilities that need to be calculated. The probability can be obtained by multiplying the probability of the variables, namely $\prod_{i=1}^{k} P(m_i)$ since they are independent. $P(m_i)$ represents the probability of the node m_i. In the same way, $P(m_i)$ can be derived by computing the conditional probability in terms of its parent nodes. The computation is repeated until a root node of the alternative network is reached.

The formula (8.4) presents the computation of conditional probabilities in the usual sense. However, most of the transaction data is encrypted during transmission. Unlike the general messages, an encrypted message needs an encryption key when coding and requires a decryption key when decoding. The difference is that cryptographic keys are required as well as the secure messages. In particular, the conditional probabilities of the nodes of pairwise keys are only relevant to the principal's neighbours. Thus, we have the following formulae to calculate the conditional probabilities of nodes of encryption or decryption, respectively.

$$P(\{m\}_k|m, k) = \frac{P(\{m\}_k) * P(m, k|\{m\}_k)}{P(m, k)} \tag{7.4}$$

In the formula (7.5), $P(m, k)$ can be obtained by multiplying $P(m)$ by $P(k)$; $P(\{m\}_k)$ represents the probability that the cipher text $\{m\}_k$ is broken; and $P(m, k|\{m\}_k)$ can be obtained by computing the conditional probability that the principals who know $E(m, k)$ also know the key k and the message m. This formula comes from the encryption rule $m \wedge k \rightarrow \{m\}_k$.

The decryption is a reverse procedure in contrast to encryption. If an intruder wants to know the plaintext of an encrypted message m, he/she must know the correct cryptographic key. Thus, we have

$$P(m|\{m\}_k, k^{-1}) = \frac{P(\{m\}_k, k^{-1}|m) * P(m)}{P(\{m\}_k, k^{-1})} \tag{7.5}$$

where k^{-1} represents the matching secret key of k.

In the formula (8.6), $P(\{m\}_k, k^{-1})$ cannot be derived by simply multiplying $P(\{m\}_k)$ by $P(k)$ because $\{m\}_k$ and k^{-1} are not independent. It can be obtained by computing the probability that the principals who know both k^{-1}. $P(\{m\}_k, k^{-1})$ represents the conditional probability that the intruder

who knows m also knows $\{m\}_k$ and k^{-1}. There are two options for the intruder to know $\{m\}_k$ and k^{-1}. One is the intruder knows $\{m\}_k$ by obtaining k and m together, and the other way is the intruder knows $\{m\}_k$ but has no knowledge about m and k at all.

Consider a BN including n nodes, Y_1 to Y_n, taken in that order. A particular value in the joint probability distribution is represented by $P(Y_1 = y_1, Y_2 = y_2, \cdots, Y_n = y_n)$, or more compactly, $P(y_1, y_2, \cdots, y_n)$. In addition, the value of any articular node is conditional only on the values of its parent nodes according to the structure of a BN. Based on the chain rule of probability theory, we thus have

$$P(x_1, x_2, \cdots, x_n) = \prod_i P(x_i | Parents(x_i)) \tag{7.6}$$

For example, by examining Figure 7.1, we can reduce its joint probability expressions.

$$\begin{aligned} P(\{m_1, m_2\}_{k_1} &= R \wedge k_1^{-1} = R \wedge \{m_1, m_2 = R \wedge m_1 = NR \wedge m_2 = R\}) \\ &= P(\{m_1, m_2\}_{k_1} = R | k_1^{-1} = R, \{m_1, m_2 = R) \\ &\quad * P(m_1, m_2 = R | m_1 = NR, m_2 = R) \end{aligned}$$

The above described the computation of conditional probabilities of variables in the network, in which a secure message transmitted in a transaction T may be linked by a number of connected nodes in DAG. The derived network can be used to reason about the domain. It can be conditioned upon any subset of their variables and supports any direction of reasoning when we observe the value of some variables. In other words, the measure of collusion attacks can be transferred to qualify the dependencies between connected nodes.

7.4.4 Experiments

Data Derivation. The experiment uses a simulated data set[1] of an electronic transaction, which consists of a message distributed over a group of principals. Each principal has a set of secure messages including cryptographic keys. As mentioned above, a cryptographic key may be a symmetric key or a pairwise key. A principal X may use one common public key to communicate with all principals, or use different keys to communicate with different principals. For simplicity, we assume that X uses only a registered public key in the transaction. Nevertheless, only the principals who are authorized to use this public key can communicate with X.

Although the cryptographic algorithms are assumed to be sound, an intruder can obtain cryptographic keys in an illegal way. Consequently, if an

[1] http://www.deakin.edu.au/∼qifengch/data2.txt

intruder obtains enough messages in collusion with a certain number of dishonest principals, he/she would have ample opportunity to uncover a secret.

It is important to determine the possibility that the messages are obtained. Table 7.3 presents an instance of the data in, in which k_i indicates symmetric keys. The variables take two values, namely known and unknown, which are replaced by 1 and 0, respectively in the real data set. The format of the data this system accepts is simple text file, in which each row contains the data corresponding to a principal. The fields in data rows should be separated by tabulators. Each column in the data file includes the discrete value corresponding to the attribute.

Table 7.3. Messages of principals in a transaction T.

M	m_1	m_2	m_3	m_4	k_1	k_2
P_1	*known*	*known*	*unknown*	*unknown*	*known*	*unknown*
P_2	*known*	*unknown*	*known*	*unknown*	*known*	*known*
P_3	*known*	*known*	*known*	*unknown*	*known*	*known*

Analysis. We used B-Course[2], a web-based data analysis tool for Bayesian modelling, to build a dependency model and discover interesting relations out of the data set. The data set can be uploaded online using the D-trail of B-Course. The format of the data this system accepts is simple text file, in which each row contains the data corresponding to each principal. The fields in data rows should be separated by tabulators.

There are 32 cases in the data file, each of which had 19 variables. B-Course allows users to control which variables are included in the analysis. k_2, k_3, K_1, K_2, K_5, K_6, K_7 and K_8 are excluded since they appear to be irrelevant to this transaction. Thus, 11 variables are considered to construct the dependency model. We continue to monitor the status of the search, and finally stop the search when the continuing search does not seem to result in better result. The most probable model for the data set is shown in the Figure 7.3, in which the arc (dependency) is measured by observing how much the probability of the model is changed by removing the arc. If the removed arc makes the model less probable, it can be viewed as a strong dependency; otherwise a weak dependency. Below is a list of selected statements describing how removing an arc affects the probability of the model. The remaining dependencies can be seen at http://www.deakin.edu.au/∼qifengch/dependence.doc. They are classified into three categories in terms of strong dependency, weak dependency and very weak dependency by using two heads right arrow, right arrow and dash right arrow, respectively.

[2] http://b-course.cs.helsinki.fi/

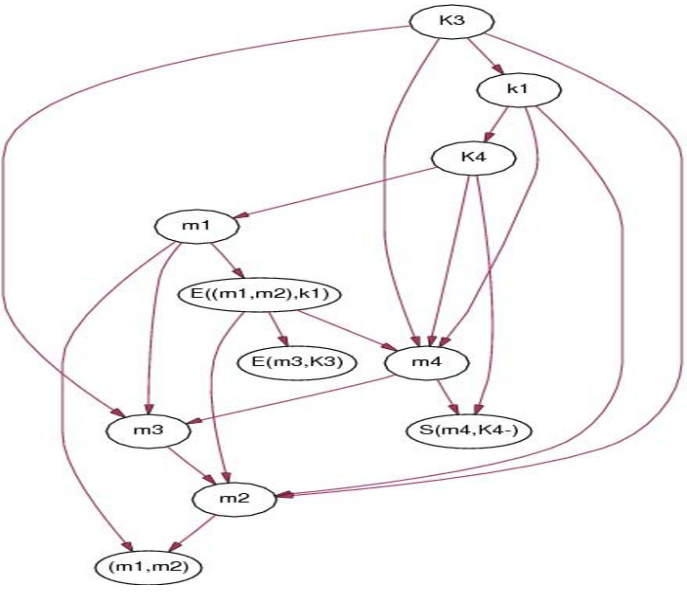

Fig. 7.3. A dependency model of the data set

Table 7.4 shows the dependencies and the ratios of the probability. For example, removing any of the strong arcs would result in a model with probability less than one-millionth of that of the original model and removing any of the weak arcs from the chosen model would decrease the probability of the model to less than one-thousandth of the probability of the original model.

Table 7.4. Strength of dependencies.

ID	Dependency	Ratio of the probability
1	$m_1 \twoheadrightarrow (m_1, m_2)$	1 : one millionth
2	$K_3 \rightarrow m_3$	1 : 5965
3	$m_2 \dashrightarrow (m_1, m_2)$	1 : 425
4	$E((m_1, m_2), k_1) \dashrightarrow m_2$	1 : 306
5	$m_4 \dashrightarrow$ S(m4, K4)	1 : 231
6	$K_4 \dashrightarrow m_4$	1 : 117
7	$m_1 \dashrightarrow E((m_1, m_2), k_1)$	1 : 12
8	$K_4 \dashrightarrow S(m_4, K_4)$	1 : 5.03

Looking at the dependency 1 and dependency 3, the former is a strong arc in comparison with the latter. Although the dependency 3 is a very weak arc, it still shows the dependence between m_2 and (m_1, m_2). The very weak dependency 3 may be due to the altered or missing m_2. Thus, the collusion

attack on the combination of m_1 and m_2 may happen if m_1 and m_2 are shared by multiple principals.

Looking at dependency 4 and the dependency 7, the former shows a dependency from cipher text to plain text, whereas the latter is on the opposite way. In conjunction with dependency 1 and dependency 3 above, they can lead to a chained dependency containing cipher text and plain text. This provides varied ways for the intruder to make collusion attacks from any place in the chain. For example, the intruder can know $E((m_1, m_2), k_1)$ by either obtaining m_1, m_2 and k_1 or obtaining $E((m_1, m_2), k_1)$ directly.

Looking at dependency 2, this is a weak dependency. It indicates that m_3 is dependent on K_3. Thus, if K_3 is known by the intruder, the intruder has an opportunity to know m_3. Nevertheless, the intruder needs to obtain the cipher text of m_3.

Looking at dependency 5, dependency 6 and dependency 8, they are actually relevant to the encryption and decryption of m_4. In the same manner, they can create a chained dependency starting from m_4, via $S(m_4, K_4)$ and K_4, to m_4. Any variable in the chain can be used by the intruder to conduct a collusion attack. For example, the intruder can know m_4 either by obtaining $S(m_4, K_4)$ and K_4 or by obtaining m_4 directly.

7.5 Summary

As we know, security protocols have played a nontrivial role in guaranteeing secure e-commerce. Also, the protocols may be subject to subtle flaws due to incomplete or vague specification. It is necessary to discover these flaws in the initial design stage of protocols. Thus, traditional formal methods have been widely used for protocol analysis.

As we have seen in the previous chapters, considerable formal approaches have been developed to check the correctness and performance of the protocols. Although they have been successfully used to identify some subtle flaws from the protocols, they have shown their limitations owing to varied and increasingly complex protocols. On the other hand, they focused on the protocol analysis in the design stage of protocols, whereas they did not attempt to evaluate the performance of protocol in a practical situation. New methods must be developed for discovering hidden but harmful attacks by utilizing the valuable transaction data from principals.

Collusion attack has been recognized as a key issue in e-commerce systems and increasingly has featured in the literatures on information security. Unfortunately, the previous formal methods have largely ignored the hidden and hazardous attack due to its complexity and the difficulty of measuring it. Nor did they demonstrate how to model it. We can only find some efforts to ensure the digital data is secure in the context of collusion. However, they may still be too expensive and hard to use.

In this chapter, we have constructed new methods for identifying collusion attacks and measuring the probability of the attacks. Our approach is a novel one because

- a novel data-mining based framework is presented to detect collusion attacks in security protocols. Especially, the set of secure messages of each principal is viewed as a transaction database. Consequently, the detection of collusion attack can be converted to identify frequent itemsets in transaction databases and to search for matching rules and facts from the available knowledge base;
- a novel, compact and intuitive Bayesian network-based scheme rather than previous key predistribution scheme is proposed to measure the probability of collusion attack. It assists in identifying the secure messages that have a high risk of causing the attack, and establishes a dependency model using a directed acyclic graph (DAG), which provides a numerical estimation of the dependencies.

To sum up, the experimental results demonstrate that the proposed novel methods in this chapter are able to assist in identifying collusion attack, correctly measuring its probability, and especially enhancing and complementing the existing protocol analysis.

8

Conclusion and Future Works

In this book, we have given a brief history and survey of the state of the art in the field of the formal methods of security protocol analysis and presented many potential directions in which it could be extended. The history of the application of formal methods to security protocol analysis spans over twenty years, dating back to the 1980s. In recent years, these methods have been showing their maturity and consolidation. A number of specialized or general-purpose tools have been developed and applied to realistic protocols, in many cases providing feedbacks to protocol designers that can be used to enhance the protocol's security. Nonetheless, we have to acknowledge that some new challenges, such as new and complex applications of the protocol and new types of threats, bring forward new requirements to existing protocol analysis. Any attempts to develop a method to ensure correctness of protocols must taken them into account. These remain some critical issues that need to be explored for improving the protocol's performance.

In this chapter these issues are outlined as emerging trends to be seen and possible future problems to be solved. In Section 8.1, we give a brief summary to the previous eight chapters. In Section 8.2, we describe some of the emerging research areas and challenging problems in protocol analysis.

8.1 Conclusion

We have introduced the application of fundamental formal methods to security protocol analysis. We start from the traditional protocol analysis, and move to new and useful methods we have developed for dealing with varied threats in new applications of security protocols. The key points are:

1. We have provided relevant preliminaries on formal methods of security protocol analysis in Chapter 1.
2. Fundamental concepts and formalism regarding formal analysis and verification of security protocols were described in Chapter 2

Q. Chen, C. Zhang, S. Zhang: Secure Transaction Protocol Analysis, LNCS 5111, pp. 217–221, 2008.

3. Chapter 3 presented a logical framework ENDL, for validating secure transaction protocols. It has some distinct features in comparison with traditional techniques, such as fail-negate, dynamics and non-monotonic. In particular, the timestamp is used for modelling the freshness of messages.
4. Chapter 4 proposed a verification model based on ENDL especially used for the analysis of electronic transaction protocols. We can write a precise definition of the behaviour of a protocol, formulate protocol properties and examine that they are satisfied via this model. These assist in overcoming the low efficiency and high rate of error in theorem proving.
5. To deal with inconsistency in secure messages, we proposed a formal framework to measure intuitively the inconsistency in secure messages in Chapter 5. It is based on the weighting majority and takes the features of secure messages into account. We aim to (1) measure the inconsistency in the messages with weights; and (2) analyse the inconsistent secure messages by evaluating their reliability. It enables the identification of the uncertain messages from the secure and insecure messages. Moreover, we presented a numerical estimation to measure the inconsistent beliefs in secure messages. A probabilistic method is used to measure intuitively the belief of different principals in terms of a minimum trust that can be put on the goal of the protocol. We then attempt to merge the inconsistent beliefs. In addition, we propose a probabilistic semantic in conjunction with ENDL, and apply the results to the protocol analysis.
6. To identify the potential correlations between secure messages for protocol analysis, we proposed a framework based on association rule mining in Chapter 6. This exempts us from predetermining authentication goals. The first phase is to collect the transaction data from principals. The second phase is to discover frequent sets of secure messages. The last phase is to identify association rules by which to evaluate the trust in the corresponding transaction.
7. In Chapter 7, we proposed new methods for identifying collusion attacks and measuring the probability of the attacks using a Bayesian network. It includes (1) identifying collusion attacks by means of matching frequent itemsets with a known knowledge base; and (2) measuring the collusion attacks using a compact and intuitive Bayesian network-based scheme. This includes determining structure, ascertaining probability variables and generating a dependency model.

Most of the methods and techniques in this book are recent work carried out by authors. In contrast to preexisting formal methods for protocol analysis, there are five positive aspects to our work.

(1) **Effectiveness in analysing electronic transaction protocols**. Our proposed ENDL is effective in verifying not only general security protocols

but also complex electronic transaction protocols. The new features of the protocols are taken into account. Furthermore, this logic can be combined with the inference engine of Prolog to enable model checking of security protocols, which can be done in a systematic way and usually faster than theorem proving.

(2) **Considering inconsistency in secure messages**. The fact that the message transmitted between principals can be inconsistent due to potential communication block, message lost and/or malicious attacks is considered. Resolving the inconsistency in secure messages before protocol analysis can enhance the reliability of verification results and ensure the correctness of the protocols.

(3) **Merging inconsistent beliefs**. The proposed techniques include probabilistic semantics to model less than perfect working conditions and draw conclusions in such cases. The belief is principals is qualified by combining assumed belief and observed belief together. The beliefs from the sender, receiver and the third party are eventually merged.

(4) **Mining inconsistent messages for protocol analysis**. The missing or inconsistent values due to a hostile environment are actually interactional. If the messages in an itemset A were lost or tampered with, this will result in the decrease of the number of occurrence of the itemset. In other words, the support and confidence of corresponding rules will decrease as well. In this regard, the protocol analysis can be transferred to identify frequent patterns.

(5) **Identifying collusion attacks and measuring their probability of occurrence**. To safeguard secure electronic transactions, collusion attacks conducted by a certain number of dishonest principals must be considered. The identification of collusion attack and evaluation of its occurrence probability are integrated into the protocol analysis we have developed. For example, to detect collusion attack, we have proposed a framework based on identifying frequent patterns and matching them with the known knowledge base.

Formal methods for security protocol analysis is a challenging and extensive field, and this book cannot range over all the issues and all the ongoing work in this area. Nonetheless, the book provides practical ways of understanding formal methods for protocol analysis, and applied them to realistic protocols. Furthermore, it presents some novel ideas to complement and enhance existing protocol analysis.

8.2 Future Work

Although formal methods of security protocol analysis have shown their success and consolidation in the past two decades, we have to acknowledge some

emerging challenges and trends in this area. With the increasing growth of transactions dependent on computer networks and security technologies, the types of applications to which a security protocol can be put become varied and complex. Moreover, the types of threats become more diverse. These issues are proposed as open problems in this section. We hope this gives us the opportunity to revisit many of these issues and find new solutions for them in the near future.

Emerging issues and trends for security protocol analysis are suggested in the following:

- developing methods to cope with the varied and complex applications to which security protocols can be put.
- proposing methods to defend against new types of threat.
- developing methods for anonymity protocol analysis.
- considering the composition problem for security protocols: given that two or more different protocols are executing under the same environment, it is possible that a message or messages from one protocol could be used to disturb the goals of the other.

We describe the above issues in more detail here:

Firstly, one of the most apparent trends is the increasingly different environments, in which protocols must interoperate and the varied applications to which a security protocol can be put. Examples may include financial transactions, which depend on novel properties such as liveness and fairness, as well as traditional security properties like integrity and confidentiality; and secure group communication, which demands a key to be kept secret within a group as participants may join or leave. As computer networks become more widespread, different platforms must interoperate. For example, we may see protocols such as Internet Key Exchange (IKE) protocol, that must agree upon both encryption keys and the used cryptographic algorithms, or SET protocol that must be capable of dealing with various credit card transactions. To meet these challenging issues, one option is to increase the complexity of the protocol. As a result, this can make the verification of security protocols become more difficult. However, it is an unavoidable tendency and will eventually have to be met at least part of way, by anybody who is interested in performing any type of security protocol analysis.

Secondly, most of the previous protocol analysis has focused on attacks in which there would be clear gain for the attacker, such as fraud or compromise of secrets. Recently, many other types of attacks are found to be related to denial of services. This requires commitment of resources, and decision of how much resources to commit, and when. It is an arduous problem and successful analysis may rely to some extent on the ability to compare the resources consumed by an attacker to the resources consumed by a defender. Current protocol analysis tools have many features that could be applied to

the problem if adapted appropriately, such as the specification of intermediate goals as well as ultimate goals. Another threat is traffic analysis. It is possible for an intruder to learn unhidden sources and destinations of message traffic, from this alone, even when encryption is used. Although a number of efforts have been applied to solve this problem, it is not easy to evaluate what amount and kind of security they provide without the ability to measure and compare the degree of protection offered by these systems. Statistical analysis can be used in this case by estimating the statistical information about source and destination that an opponent could learn by observing or interfering with message traffic.

Thirdly, in a simple anonymous commutation, a user commits a request via a single site, which removes the request of identifying data or otherwise hides its destination, and forwards it to the server. The anonymity is not obvious in isolation. It would be difficult to disguise the source of a request if it is the only request in the network. Actually, anonymizing protocols rely on a mixture of traffic to achieve this goal. Statistical analysis will be useful to determine how well this strategy performs in different situations. Nonetheless, it is unrealistic for an attacker to break the protocols without communicating with other principals. Thus, we should focus on a certain number of principals who are assumed to share knowledge.

In the end, most work on the application of formal methods for security protocol analysis has focused on the analysis of protocols that can be described according to a single sequence of messages without any option points or loops. In reality, many security protocols as they are executed can be viewed as a suite of sub-protocols in conjunction with a collection of option points, in which the user can choose which sub-protocol to implement. A message from a protocol could be used in the other protocols. It may be very difficult to confirm that no protocol in the collection will accept a message from another protocol in the collection. Nonetheless, one realistic way is to reduce the number of state transitions that had to be examined whenever we had to determine how a message could be produced. On the other hand, two messages can be confused with each other. In that case, it might be useful to determine where such confusion is likely to happen.

Although there are still many other interesting problems in the field of formal methods for protocol analysis, in our point of view, the emerging issues and trends listed are essential, they require extra attention and need to be treated carefully.

References

1. Abadi, M.: Secret by typing in security protocols. Journal of the ACM 46(5), 749–786 (1999)
2. Abadi, M., Tuttle, M.: A semantics for a logic of authentication. In: Proceedings of the 10th Symposium on Principles of Distributed Computing, pp. 201–216 (1991)
3. Agrawal, R., Imielinshki, T., Swami, A.: Mining Association Rules between Sets of Items in Large Databases. In: Proceeding of ACM-SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
4. Agrawal, R., Srikant, R.: Privacy-Preserving Data Mining. In: Proceeding of the ACM SIGMOD Conference on Management of Data, pp. 439–450 (2000)
5. Alex, G., Ehud, G.: Privacy preserving Data Mining Algorithms without the use of Secure Computation or Perturbation. In: Proceedings of the 10th International Database Engineering and Applications Symposium, pp. 121–128. IEEE Computer Society Press, Washington (2006)
6. Bai, S., Sui, L.Y., Chen, Q.F., Fu, Y., Zhuang, C.: The verification logic for secure electronic protocols. Journal of Software 11(2), 213–221 (2002)
7. Bella, G., Massacci, F., Paulson, L., Tramontano, P.: Formal Verification of Cardholder Registration in SET. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 159–174. Springer, Heidelberg (2000)
8. Birrell, A.: Secure Communications Using Remote Procedure Calls. ACM Transaction on Computer Systems 3(1), 1–14 (1985)
9. Bolignano, D.: An approach to the formal verification of cryptographic protocols. In: Proceedings of the Third ACM Conference on Computer and Communications Security, pp. 106–118. ACM Press, New York (1996)
10. Bolignano, D.: Towards the Formal Verification of Electronic Commerce Protocols. In: Proceedings of the IEEE Computer Security Foundations Workshop X, pp. 133–146. IEEE Computer Society Press, Los Alamitos (1997)
11. Boneh, D., Shaw, J.: Collusion-secure fingerprinting for digital data. IEEE Transactions on Information Theory 44(5), 1897–1905 (1998)
12. Bonnie, B.A., James, V.H., Paul, B.L., Scott, L.: Standards and verification for fair-exchange and atomicity in e-commerce transactions. Inf. Sci. 176(8), 1045–1066 (2006)
13. Boyd, C., Mathuria, A.: Protocols for key establishment and authentication. Springer, London (2002)

14. Brackin, S.: A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. In: Proceedings of the 1996 IEEE Computer Security Foundations Workshop IX, pp. 62–76. IEEE Computer Society Press, Los Alamitos (1996)
15. Brackin, S.: A State-Based HOL Theory of Protocol Failure, *ATR 98007*. Arca Systems, Inc. (1997), `http://www.arca.com/paper.htm`
16. Brackin, S.: An Interface Specification Language for Automatically Analyzing Cryptographic Protocols. In: Proceedings of the 1997 Symposium on Network and Distributed System Security, pp. 40–51. IEEE Computer Society Press, Los Alamitos (1997)
17. Brackin, S.: Automatic formal analyses of two large commercial protocols. In: Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols (1997)
18. Brackin, S.: Evaluating and improving protocol analysis by automatic proof. In: Proceedings of the 11th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos (1998)
19. Brands, S.: Restrictive blinding of secret-key certificates. Technical Report CS-R9509, CWI-Certrum voor Wiskunde en Informatica (1995)
20. Bratko, I.: Prolog Programming for Artificial Intelligence. Addison-Wesley, Reading (1990)
21. Burns, J., Mitchell, C.: A Security Scheme for Resource Sharing over a Network. Computers and Security 19, 67–76 (1990)
22. Burrows, M., Abadi, M., Needham, R.: A logic for Authentication. ACM Transactions on Computer Systems 8(1), 18–36 (1990)
23. Caelli, W., Longley, D., Shain, M.: Information security handbook. Macmillan, Basingstoke (1994)
24. Campbell, E.: Safavi-Naini R., and Pleasants P.A., Partial Belief and Probabilistic Reasoning in the Analysis of Secure Protocols. In: Proceedings. Computer Security Foundations Workshop V, pp. 84–91. IEEE Computer Society Press, Los Alamitos (1992)
25. Celik, M.U., Sharma, G., Tekalp, A.: Collusion-resilient fingerprinting using random pre-warping. In: Proceeding of IEEE International Conference of Image Processing, pp. 509–512 (2003)
26. Chaum, D.: Untraceable electronic mail, return addresses and digital signatures. Communications of the ACM 24(2), 84–90 (1981)
27. Chen, Q.F., Zhang, C.: The Verification logic for Secure Electronic Transaction Protocols. In: Proceedings of the 2002 Intelligent Information Technology, pp. 326–333. Beijing, China (2002)
28. Chen, Q.F., Zhang, C.Q., Zhang, S.: A Verification Model for Electronic Transaction Protocols. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 824–833. Springer, Heidelberg (2004)
29. Chen, Q.F., Zhang, C.Q., Zhang, S.: ENDL: A Logical Framework for Verifying Secure Transaction Protocols. Knowledge and Information Systems 7(1), 84–109 (2005)
30. Chen, Q.F., Zhang, C.Q., Zhang, S.C., Li, C.: Verifying the Purchase Request in SET Protocol. In: Zhou, X., Zhang, Y., Orlowska, M.E. (eds.) APWeb 2003. LNCS, vol. 2642, pp. 263–274. Springer, Heidelberg (2003)
31. Chen, Q.F., Zhang, S.: Dealing with Inconsistent Secure Messages. In: Zhang, C., W. Guesgen, H., Yeap, W.-K. (eds.) PRICAI 2004. LNCS (LNAI), vol. 3157, pp. 33–42. Springer, Heidelberg (2004)

32. Chen, Q.F., Chen, Y.: Analyzing Security Protocols Using Association Rule Mining. In: Zhang, S., Jarvis, R. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 245–253. Springer, Heidelberg (2005)
33. Chen, Q.F., Chen, Y.Y.P., Zhang, C.Q., Zhang, S.: A Framework for Merging Inconsistent Beliefs in Security Protocol Analysis. In: Proceedings of DEEC 2005, pp. 119–124. IEEE Computer Society Press, Los Alamitos (2005)
34. Chen, Q.F., Chen, Y.P.P., Zhang, S.C., Zhang, C.: Detecting Collusion Attacks in Security Protocols. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 297–306. Springer, Heidelberg (2006)
35. Chen, Q.F., Chen, Y.P.P., Zhang, S.: Identifying Hidden Patterns in Secure Message for Protocol Analysis. In: Zhang, Z., Siekmann, J.H. (eds.) KSEM 2007. LNCS (LNAI), vol. 4798, pp. 30–38. Springer, Heidelberg (2007)
36. Cheung, Y.L., Fu, A.: Mining Frequent Itemsets without Support Threshold: With and Without Item Constraints. IEEE Transaction on Knowledge and Data Engineering 16(9), 1052–1069 (2004)
37. Clarke, E.M., Wing, J.: Formal methods: State of the art and future directions. ACM Computing Surveys 28(4), 626–643 (1996)
38. Cox, B., Tygar, J., Sirbu, M.: Netbill security and transaction protocol. In: Proceedings of the First USENIX Workshop in Electronic Commerce (1995)
39. Dalal, M.: Investigations into A Theort of Knowledge Base Revision: Preliminary Report. In: Proceedings of AAAI-88, pp. 475–479 (1988)
40. Denning, D., Sacco, G.: Timestamp in Key Distribution Protocols. Communications of ACM 24(8), 533–536 (1981)
41. Dierks, T., Allen, C.: `http://www.ietf.org/rfc/rfc2246.txt` (1999)
42. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory IT-22, 644–654 (1976)
43. Dolev, D., Even, S., Karp, R.: On the security of ping-pond protocols. Information and Control, 57–68 (1982)
44. Dolev, D., Yao, A.: On the Security of Public Key Protocols. IEEE Transaction on Information Theory 29(2), 198–208 (1983)
45. Dowek, G., Felty, A., Herbelin, H., Huet, G., Murthy, C., Parent, C., Paulin-Mohring, C., Werner, B.: The coq proof assistant user guide. Rapport INRIA 154 (1993)
46. Du, W., Deng, J., Han, Y., Varshney, P., Katz, J., Khalili, A.: A pairwise key predistribution scheme for wireless sensor networks. ACM Transactions on Information and System Security 8(2), 228–258 (2005)
47. Ettinger, J.: Information security. Chapman & Hall, Boca Raton (1993)
48. Farr, M., Chadwick, B., Wong, K.: Security for computer systems. National Computing Centre, Manchester (1972)
49. Fabrega, F., Herzog, J., Guttman, J.: Honest Ideals on Strand Spaces. In: Proceedings of the IEEE Computer Security Foundations Workshop XI, pp. 66–77. IEEE Computer Society Press, Los Alamitos (1998)
50. Fabrega, F., Herzog, J., Guttman, J.: Strand Spaces: Why is a security protocol correct. In: Proceedings of the 1998 IEEE Symposium on Security and Privacy, pp. 160–171. IEEE Computer Society Press, Los Alamitos (1998)
51. Fagin, R., Halpern, J.: Uncertainty, Belief and Probability. In: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence 2, pp. 1161–1167 (1989)

52. Fernández, M., Soriano, B.M.: Fingerprinting concatenated codes with efficient identification. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 459–470. Springer, Heidelberg (2002)
53. Formal Systems (Europe) Ltd.: Failuers-Divergence Refinement, FDR2 User Manual, `http://www.fsel.com/`
54. Frier, A., Karlton, P., Kocher, P.: The SSL 3.0 Protocol. Netscape Communications Corp. (1996), `http://home.netscape.com/eng/ssl3/ssl.toc.html`
55. Gabber, E., Silberschatz, A.: Agora: a Minimal Distributed Protocol for Electronic Commerce. In: Proceedings of the 2 nd USENIX Workshop on Electronic Commerce, California, pp. 223–232 (1996)
56. Gardenfors, P.: Knowledge in Flux. MIT Press, Cambridge (1998)
57. Gavin, L.: Casper - A compiler for the analysis of security protocols, User Manual and Tutorial, Ver1.3 (1999)
58. Gerard, J.: The Model Checker SPIN. IEEE Transactions on Software Engineering 23(5), 279–295 (1997)
59. Ghosh, A.: Security and privacy for E-business. Wiley, Chichester (2001)
60. Gong, L.: Handling infeasible specifications of cryptographic protocols. In: Proceedings of Computer Security Foundations Workshop IV, Franconia NH, pp. 99–102 (1991)
61. Gong, L.: A Security Risk of Depending on Synchronized Clocks. ACM Operating Systems Review 26(1), 49–53 (1992)
62. Gong, L., Needham, R., Yahalom, R.: Reasoning about belief in cryptographic protocols. In: Proceedings of the Symposium on Security and Privacy, Oakland, CA, pp. 234–248 (1990)
63. Gong, L., Syverson, P.: Fail-Stop Protocols: An Approach to Designing Secure Protocols. In: 5^{th} International Working Conference on Dependable Computing for Critical Applications, pp. 44–55 (1995)
64. Goralski, W.: TCP/IP applications and protocols. Computer Technology Research Corp., Charleston (1995)
65. Gritzalis, S.: Security Protocols over open networks and distributed systems: Formal methods for their Analysis, Design, and Verification. Computer Communications 22(8), 695–707 (1999)
66. Halpern, J.Y., Fagin, R.: Two views of belief: belief as generalized probability and belief as evidence. Artificial Intelligence 54, 275–317 (1992)
67. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1–12 (2000)
68. Heintze, N., Tygar, J., Wing, J., Wong, H.: Model Checking Electronic Commerce Protocols. In: Proceedings of the 2nd USENIX Workshop on Electronic Commerce (1996)
69. Holzmann, G.: Design and Validation of Computer Protocols. Prentice-Hall, Englewood Cliffs (1991)
70. Huberman, B., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of ACM electronic commerce99, pp. 78–86 (1999)
71. Hunter, A.: Measuring Inconsistency in Knowledge via Quasi-classical Models. In: Proceedings of AAAI '02, pp. 68–73 (2002)
72. Hunter, A.: Evaluating the Significance of Inconsistencies. In: Proceedings of the International Joint Conference on AI (IJCAI'03), pp. 468–473 (2003)

73. http://www.unctad.org/en/docs/ecdr2002_en.pdf (2002)
74. Indrakshi, R.: An anonymous fair exchange e-commerce protocol. In: Proceeding of the First International Workshop on Internet Computing and E-commerce, San Francisco, CA (2001)
75. Indrakshi, R., Indrajit, R.: Failure Analysis of an E-commerce Protocol Using Model Checking. In: Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, Milpitas, CA, pp. 176–183 (2000)
76. Indrajit, R., Indrakshi, R., Natarajan, N.: A Fair-exchange E-commerce Protocol with Automated Dispute Resolution. In: Proceedings of the IFIP TC11/WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions, pp. 27–38 (1999)
77. Inggs, C.P., Barringer, H.: CTL ast Model Checking on a Shared-Memory Architecture. Electr. Notes Theor. Comput. Sci. 128(3), 107–123 (2005)
78. Inggs, C., Barringer, H., Nenadic, A., Zhang, N.: Model Checking a Security Protocol. In: Proceedings of the Southern African Telecommunications Network and Applications Conference – SATNAC '04, South Western Cape, South Africa (2004)
79. ITU-T: ITU-T X.509, The Directory - An Authentication Framework, ITU-T (1998)
80. Jakobsson, M.: Mix-Based Electronic Payments. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 157–173. Springer, Heidelberg (1999)
81. Karn, P., Simpson, W.: Photuris Session-key management protocol, RFC 2522. Internet Engineering Task Force (1999)
82. Kailar, R.: Reasoning about Accountability in Protocols for Electronic Commerce. In: Proceedings of the 1995 IEEE Symposium on Security and Privacy, pp. 236–250. IEEE Computer Society Press, Los Alamitos (1995)
83. Kemmerer, R.: Analyzing encryption protocols using formal verification techniques. IEEE Journal on Selected Areas in Communications 7(4), 448–457 (1989)
84. Kemmerer, R., Meadows, C., Millen, J.: Three Systems for Cryptographic Protocol Analysis. Journal of Cryptology 7(2), 79–130 (1994)
85. Kessler, V., Neumann, H.: A Sound Logic for Analysing Electronic Commerce Protocols. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 345–360. Springer, Heidelberg (1998)
86. Kessler, V., Wedel, G.: AUTLOG-An advanced Logic of Authentication. In: Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII, pp. 90–99. IEEE Computer Society Press, Los Alamitos (1994)
87. Kohl, J., Neuman, C.: The Kerberos Network Authentication Service, Version 5 RFC, Draft No. 4. Network Working Group, MIT Project Athena (1990)
88. Kong, W., Ogata, K., Xiang, J., Futatsugi, K.: Formal analysis of an anonymous fair exchange e-commerce protocol. In: Proceeding of the Fourth International Conference on Computer and Information Technology, pp. 1100–1107. IEEE Computer Society Press, Los Alamitos (2004)
89. Konieczny, S., Pérez, R.: Merging Information Under Constraints. A Logical Framework, Journal of Logic and Computation 12(5), 773–808 (2002)
90. Konieczny, S., Pérez, R.: On the Frontier between Arbitration and Majority. In: Proceedings of the Eigth International Conference on Principles of Knowledge Representation and Reasoning, pp. 109–118 (2002)

91. Lee, W.K., Stolfo, S., Mok, K.: Mining Audit Data to Build Intrusion Detection Models. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98), pp. 66–72 (1998)
92. Leonard, N.: A security architecture for multi-agent matchmaking. In: Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS96), pp. 80–86. AAAI Press, Menlo Park (1996)
93. Liberatore, P., Schaerf, M.: Arbitration (or How to Merge Knowledge Bases). IEEE Transaction on Knowledge and Data Engineering 10(1), 76–90 (1998)
94. Liebl, A.: Authentication in Distributed Systems: A bibliography. ACM Operating Systems Review 27(4), 31–41 (1993)
95. Lin, J.: Integration of Weighted Knowledge Bases. Artificial Intelligence 83(2), 363–378 (1996)
96. Lin, J., Mendelzon, A.: Knowledge base merging by majority. In: In Dynamic Worlds: From the Frame Problem to Knowledge Management, Kluwer Academic Publishers, Dordrecht (1999)
97. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
98. Lowe, G.: Some new attacks upon security protocols. In: Proceedings of Computer Security Foundations Workshop (CSFW96), pp. 139–146. IEEE Computer Society Press, Los Alamitos (1996)
99. Lowe, G.: Casper: A Compiler for the Analysis of Security Protocols. In: Proceedings of the 1997 IEEE Computer Security Foundations Workshop X, pp. 18–30. IEEE Computer Society Press, Los Alamitos (1997)
100. Lu, S., Smolka, S.: Model Checking the Secure Electronic Transaction (SET) Protocol. In: Proceedings of 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 358–365 (1999)
101. Ma, X.Q., Cheng, X.C., McCrindle, R.: Knowledge Based Approach for Mechanically Verifying Security Protocols. In: Proceeding of IJCAI 2005, pp. 1572–1573 (2005)
102. Maiwald, E.: Network security: a beginner's guide, Berkeley, California. Osborne/McGraw-Hill, London (2001)
103. Maggi, P., Sisto, R.: Using SPIN to Verify Security Properties of Cryptographic Protocols. In: Bošnački, D., Leue, S. (eds.) SPIN 2002. LNCS, vol. 2318, p. 187. Springer, Heidelberg (2002)
104. Marcus, G.: Firewalls: a complete guide. McGraw-Hill, London (2000)
105. Martínez-Ballesté, A., Sebé, F., Domingo-Ferrer, J., Soriano, M.: Practical Asymmetric Fingerprinting with a TTP. In: Proceedings of the 14th International Workshop on Database and Expert Systems Applications, pp. 352–357 (2003)
106. Maurer, U.: Modelling a Public-Key Infrastructure. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 324–350. Springer, Heidelberg (1996)
107. Markowitch, O., Gollmann, D., Kremer, S.: On Fairness in Exchange Protocols. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 451–464. Springer, Heidelberg (2003)
108. McMillan, K.: Symbolic model checking: An approach to the state explosion problem. Technical Report CMU-CS-92-131, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, Ph.D thesis (1992)

109. Meadows, C.: Applying Formal Methods to the Analysis of a Key-Management Protocol. Journal of Computer Security 1, 5–35 (1992)
110. Meadows, C.: Language Generation and Verification in the NRL Protocol Analyzer. In: Proceedings of the IEEE Computer Security Foundations Workshop IX, pp. 48–61. IEEE Computer Society Press, Los Alamitos (1996)
111. Meadows, C.: The NRL Protocol Analyzer: An overview. Journal of Logic Programming 26(2), 113–131 (1996)
112. Meadows, C.: Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. IEEE Journal on Selected Areas in Communication 21(1), 44–54 (2003)
113. Meadows, C., Syverson, P.: A formal specification of requirements for payment transactions in the SET protocol. In: Hirschfeld, R. (ed.) FC 1998. LNCS, vol. 1465, pp. 122–140. Springer, Heidelberg (1998)
114. Millen, J.: The Interrogator Model, Proceedings of the 1995. In: IEEE Symposium on Security and Privacy, pp. 251–260. IEEE Computer Society Press, Los Alamitos (1995)
115. Millen, J.: CAPSL - Common Authentication Protocol Specification Language (1997), `http://www.mitre.org/research/capsl`
116. Millen, J., Neuman, C., Schiller, J., Saltzer, J.: Kerberos Authentication and Authorization system, Project Athena Technical Plan, Section E.2.1. M.I.T., MA (1987)
117. Moser, L.: A Logic of Knowledge and Belief for Reasoning about Computer Security. In: Proceedings of the Computer Security Foundations Worksshop II, Washington, pp. 57–63 (1989)
118. Murdoch, S.J., Danezis, G.: Low-Cost Traffic Analysis of Tor. In: IEEE Symposium on Security and Privacy 2005, pp. 183–195 (2001)
119. Neapolitan, R.E.: Learning Bayesian networks. Prentice Hall, Englewood Cliffs (2004)
120. Needham, R., Schroeder, M.: Using Encryption for Authentication in Large Networks of Computers. Comm. of the ACM 21(12), 993–999 (1978)
121. Nessett, D.: A Critique of the BAN Logic. ACM Operating Systems Review 24(2), 35–38 (1990)
122. Netscape (Netscape), `http://wp.netscape.com/eng/ssl3/`
123. Neuman, B.: Neuman B, Ts′o T.: Kerberos: An Authentication Service for Computer Networks. IEEE Communications 32(9), 33–38 (1994)
124. Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
125. Otway, D., Rees, O.: Efficient and timely mutual authentication. ACM Operating Systems Review 21(1), 8–10 (1987)
126. Panti, M., Spalazzi, L., Tacconi, S.: Using the NuSMV Model Checker to verify the Kerberos Protocol. In: Proceedings of the Third Collaborative Technologies Symposium (CTS-02), pp. 220–230. San Antonio, Texas (2002)
127. Papa, M., Bremer, O., Hale, J., Shenoi, S.: Formal analysis of e-commerce protocols. In: Proceedings of the 5th International Symposium on Autonomous Decentralized Systems, pp. 19–28 (1997)
128. Paulson, L.: Mechanized Proofs for a Recursive Authentication Protocol. In: Proceedings of the IEEE Computer Security Foundations Workshop X, pp. 84–94. IEEE Computer Society Press, Los Alamitos (1997)

129. Paulson, L.: Proving Properties of Security Protocols by Induction. In: Proceedings of the IEEE Computer Security Foundations Workshop X, pp. 70–83. IEEE Computer Society Press, Los Alamitos (1997)
130. Piper, F., Murphy, S.: Cryptography: A Very Short Introduction. Oxford University Press, Oxford (2002)
131. Ranagan, P.: An Axiomatic Basis of Trust in Distributed Systems. In: Proceedings of the IEEE Symposium on Security and privacy, Washington, pp. 204–211 (1998)
132. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining Digital Signatures and Public Key Cryptosystems. Comm. of ACM 21(2), 120–126 (1978)
133. Robinson, J.: A machine oriented logic based on the resolution principle. Journal of the Association for Computing Machinery 12, 23–41 (1965)
134. Robshaw, M.: MD2, MD4, MD5, SHA and other hash functions. Tech. Rep. TR-101, RSA Labs, version 4.0 (1995)
135. Schaerf, M., Cadoli, M.: Tractable Reasoning via Approximation. Artificial Intelligence 74, 249–310 (1995)
136. Scheid, J., Holtsberg, S.: Ina Jo Specification Language Reference Manual. Systems Development Group, Unisys Corporation (1988)
137. Schneider, S.: Verifying Authentication Protocols with CSP. In: Proceedings of the IEEE Computer Security Foundations Workshop X, pp. 3–17. IEEE Computer Society Press, Los Alamitos (1997)
138. SET Secure Electronic Transaction Specification, Book 1: Business Description, Version 1.0 (May 31, 1997)
139. SET Secure Electronic Transaction Specification, Book 2: Programmer's Guide, Version 1.0 (May 31, 1997)
140. SET Secure Electronic Transaction Specification, Book 3: Formal Protocol Definition, Version 1.0 (May 31, 1997)
141. Song, D.X., Wagner, D., Tian, X.: Timing analysis of keystrokes and timing attacks on SSH. In: Tenth USENIX Security Symposium (2001)
142. Stanley, R.M.: Oliveira and Osmar R. Zaïane.: Toward Standardization in Privacy-Preserving Data Mining. In: Proceeding of the 3nd Workshop on Data Mining Standards, in conjunction with KDD 2004, Seattle, WA, USA, pp. 7–17 (2004)
143. Sherif, M.: Protocols for Secure Electronic Commerce. CRC Press, Boca Raton (2000)
144. Sidhu, D.: Authentication Protocols for Computer Networks. Computer Networks and ISDN Systems 11, 297–310 (1986)
145. Simmons, G.: How to Selectively Broadcast a Secret. In: Proceedings of the 1985 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos (1985)
146. Stone H.: Analysis of attacks on image watermarks with randomised coefficients. NEC Technical Report (1996)
147. Syverson, P.: The Use of Logic in the Analysis of Cryptographic Protocols. In: Proceedings of the 1991 IEEE Computer Security Symposium on Security and Privacy, pp. 156–170. IEEE Computer Society Press, Los Alamitos (1991)
148. Syverson, P.: A Taxonomy of Replay Attacks. In: Proceedings of the 7th IEEE ComputerSecurity Foundations Workshop, pp. 131–136. IEEE Computer Society Press, Los Alamitos (1994)

149. Syverson, P., Measows, C.: A logical language for specifying cryptographic protocol requirements. In: Proceeding of the IEEE Computer Society Symposium on Research in Security and Privacy, pp. 165–177 (1993)
150. Syverson, P., van Oorschot, P.: On Unifying some Cryptographic Protocol Logics. In: Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII, pp. 14–29. IEEE Computer Society Press, Los Alamitos (1994)
151. Thomas, S.: HTTP essentials: protocols for secure, scaleable, Web sites. Wiley, New York (2001)
152. Tygar, J.: Atomicity in Electronic Eommerce. In: Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, pp. 8–26 (1996)
153. U.S. National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Draft Federal Information Processing Standards Publication 46-3, FIPS PUB 46-3 (1999)
154. Varadharajan, V.: Verification of Network Security Protocols. Computers and Security 8(8), 693–708 (1989)
155. VISA2003 3-D Secure protocol specification - Extensions for Mobile Internet Devices, International (2003), `visa.com/fb/main.jsp`
156. Wahadaniah, V., Guan, Y.L., Chua, H.C.: A New Collusion Attack and Its Performance Evaluation. In: Petitcolas, F.A.P., Kim, H.-J. (eds.) IWDW 2002. LNCS, vol. 2613, pp. 64–80. Springer, Heidelberg (2003)
157. Wei, J., Cheung, S.C.: Wang Xu.: Exploiting automatic analysis of e-commerce protocols. In: Proceeding of 25th Annual International Computer Software and Applications Conference, pp. 55–62 (2001)
158. Wing, J., Vaziri-Farahani, M.: A Case Study in Model Checking Software Systems. Science of Computer Programming 28, 273–299 (1997)
159. Wong, T., Wang, C., Wing, J.: Verifiable Secret Redistribution for archive systems. In: Proceedings of the First International IEEE Security in Storage Workshop (2002)
160. Woo, T., Lam, S.: A semantic model for authentication protocols. In: Proceedings of the IEEE Computer Sciety Symposium on Research in Security and Privacy, pp. 178–194 (1993)
161. Yu, T., Ma, X., Winslett, M.: Prunes: An Efficient and Complete Strategy for Trust Negotiation Over the Internet. In: Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-7), pp. 210–219 (2000)
162. Yurcik, W., Doss, D.: Internet Attacks: A Policy Framework for Rules of Engagement. In: Proceedings of the 29 Research Conference on Communication, Information, and Internet Policy (TPRC), Alexandria, VA, USA (2001)
163. Zhang, C., Zhang, S.: Association Rule Mining. LNCS (LNAI), vol. 2307. Springer, Heidelberg (2002)

Index

access control, 37
accumulation, 94
algebraic, 41
anonymity, 6, 31
anonymous communication, 11
association rule mining, 175, 182
assumed belief, 160
attack-construction method, 48
authentication, 27, 87
authentication protocol, 19
authenticity, 6
authorization, 29
availability, 31

backward chaining, 115
BAN logic, 7, 61
belief logic, 19

cascade protocol, 53
certificate, 35
ciphertext, 20, 33
collusion attack, 10, 193
confidentiality, 25
conflicting belief, 156
counter-example, 6
cryptanalysis, 19
CTL, 7

data authentication, 27
data mining, 175
data security, 32
denial of service, 10, 40
DES, 11
detection, 178, 193
digital signature, 36

e-commerce protocol, 2
encryption construction, 2
encryption key, 18
ENDL, 71, 86, 89, 113
entity authentication, 2
epistemic logic, 19

fact database, 115
fail-negate, 75
fairness, 10
FDR, 7, 109
fingerprint, 37, 195
finite-state, 6
formal analysis, 3
formal logic, 7
formal method, 5
formulae, 18
forward chaining, 115
FP, 201
frequent pattern, 198
freshness, 29, 30

GNY logic, 49

higher-order logic, 5

identification, 87, 198
IKE protocol, 10
inconsistency, 134
inference rule, 19
inference-construction method, 48
integration, 156
integrity, 6, 25
interleave, 40
internet protocol, 21

Isabelle, 68
itemset, 201

key agreement or establishment, 2
key authentication, 28
Kripke structure, 6

liveness, 10, 30
LTL, 7

MAC, 47
man-in-the-middle attack, 34, 38
measuring, 144
minimum support, 201
model checking, 6, 107

name-stamp protocol, 54
NDL, 85
Netbill protocol, 23
network security, 32
non-monotonic, 75
non-repudiation, 26, 30
NRL, 48, 56

observed belief, 160
one-way hash function, 35

password, 37
PKI, 35, 84
plaintext, 20, 33
principal, 18
privacy, 179
probability, 159
proof-construction method, 50
public key cryptography, 34

reflection, 41
RSA, 34

secrecy, 6
Secure data transport, 2
secure transaction protocol, 2
security protocol, 1
security service, 24
session key, 19
SET protocol, 10, 22
strong encryption, 19
symmetric cryptography, 32
system security, 32

temporal logic, 8
theorem proving, 5
timestamp, 11, 37, 86, 180
traffic analysis, 10, 41
trusted third party, 84, 86

uncertainty, 131

watermark, 195
weight, 160