Automated Security Analysis of Payment Protocols

by

Enyang Huang

B.E. The University of New South Wales (2007)
S.M. Massachusetts Institute of Technology (2010)

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in the Field of Computer Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author . . . . [illegible]
Department of Civil and Environmental Engineering
May 22, 2012

Certified by . . . . [illegible]
George A. Kocur
Senior Lecturer in Civil and Environmental Engineering
Thesis Supervisor

Accepted by . . . . [illegible]
Heidi M. Nepf
Chair, Departmental Committee for Graduate Students

Automated Security Analysis of Payment Protocols

by

Enyang Huang

Submitted to the Department of Civil and Environmental Engineering
on May 22, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in the Field of Computer Engineering

Abstract

Formal analyses have been used for payment protocol design and verification but, despite developments in semantics and expressiveness, previous literature has placed little emphasis on the automation aspects of the proof systems. This research develops an automated analysis framework for payment protocols called PTGPA. PTGPA combines the techniques of formal analysis as well as the decidability afforded by theory generation, a general-purpose framework for automated reasoning.

A comprehensive and self-contained proof system called TGPay is first developed. TGPay introduces novel developments and refinements in the formal language and inference rules that conform to the prerequisites of theory generation. These target desired properties in payment systems such as confidentiality, integrity, authentication, freshness, acknowledgement and non-repudiation. Common security primitives such as encryption, decryption, digital signatures, message digests, message authentication codes and X.509 certificates are modeled.

Using TGPay, PTGPA performs analyses of payment protocols under two scenarios in full automation. An Alpha-Scenario is one in which a candidate protocol runs in a perfect environment without attacks from any intruders. The candidate protocol is correct if and only if all pre-conditions and post-conditions are met. PTGPA models actions and knowledge sets of intruders in a second, modified protocol that represents an attack scenario. This second protocol, called a Beta-Scenario, is obtained mechanically from the original candidate protocol, by applying a set of elementary capabilities from a Dolev-Yao intruder model.

This thesis includes a number of case studies to demonstrate the feasibility and benefits of the proposed framework. Automated analyses of real-world bank card payment protocols as well as newly proposed contactless mobile payment protocols are presented. Security flaws are identified in some of the protocols; their causes and implications are addressed.

Thesis Supervisor: George A. Kocur
Title: Senior Lecturer in Civil and Environmental Engineering

Acknowledgments

I want to express my sincere gratitude to my doctoral advisor Dr. George Kocur. This thesis could not have been what it is without his advice, guidance and funding support. It has been an honor to work with him at MIT.

I must thank my amazing doctoral committee members. I am indebted to Prof. Marta González and Prof. Saurabh Amin, for their support and invaluable suggestions.

I am fortunate to have been able to work with so many distinguished faculty members during my four years' stay at MIT. I am also grateful to my colleagues and friends who have accompanied me on this fantastic journey.

I want to acknowledge Damien Balsan and Will Graylin for their motivation and inspiration for research in the domain of payment.

My gratitude goes to my family, Mom and Dad on the other side of the Pacific Ocean, my aunt and cousins here in the US. I must thank my wife, Lunan, for her constant love and support. This work is dedicated to all of you.

Last but not least, unbounded thanks to Andi Zhou from Yale University for perfecting this thesis. Any remaining errors are my own.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

Contents

1.1 Thesis Motivation

Near-Field-Communication (NFC) [NF12a] [ISO04] is a low latency, RFID [Fin03] communication technology that has been widely adopted in the proximity payment industry. NFC allows many novel payment applications. Contactless bank card trials have been conducted throughout the world [Exp05] [Mas05] [Vis07]. Mobile-enabled applications such as Google Wallet [Goo11] are being deployed and used daily by merchants and customers.

Despite the early success of NFC and related payment technologies such as EMV [EMV11a] [EMV11b], verification of modern payment protocols for correctness remains an active and challenging topic for research. Traditionally, these crucial analyses have been done by hand. Assumptions are first made based on the levels of abstraction in which protocols were specified. Properties to be achieved by the protocols are defined and then verified in an informal and ad-hoc way. The overall procedure can be very complex and tedious, and is often prone to errors arising from many possible sources. This inevitably calls for more rigorous approaches to payment protocol verification.

Formal methods, in which a given protocol is specified and structurally proved correct in a formal mathematical semantic, have much to offer. The increased overall analysis complexity can be justified by the importance of security and message integrity. In the past, a number of significant formal frameworks have been developed, among which is the notable progenitor in this domain, BAN logic [BAN89] [BAN90]. BAN logic is a light weight framework long recognized for its elegance and intuitiveness. It provides formal semantics for determining properties of authentication protocols. At the core of the BAN formalism is a proof system, consisting of a set of inference rules called axioms. Security goals and business requirements of a protocol can then be verified by applying logical deductions from applications of the axioms.

Although BAN logic was initially designed to verify authentication protocols, researchers have extended its semantics to uncover security flaws in a much broader context. Similar frameworks have since been applied to analyses of communication, key distribution and electronic commerce protocols [SvO94] [Kai96] [SM09].

Despite developments in semantics and expressiveness of BAN-style inference systems, previous literature did not emphasize the procedural mechanism in the application of the proof system. A protocol is well specified in a formal language but the proofs of its correctness are often still done manually. This approach may not be sufficient for analyses of complex protocols. Therefore, instead of deriving inference chains in an ad-hoc way with human guidance, fully automated algorithms that control exact behaviors of inference procedures need to be developed.

Various attempts were made to automate the BAN-style formalisms, particularly, [KW94] [MSnN95]. In 1999, Kindred and Wing [Kin99] [KW99] made a remarkable contribution that circumvents the problem of non-termination, a major barrier to BAN automation. Their work was based on the idea of finite theory generation, borrowing ideas from syntactic methods and saturation theory. In their method, constraints are enforced when writing rules of inference and an enumeration algorithm is developed to ensure that the proof system terminates and is both sound and complete, and therefore decidable. This provides a plausibility argument that semantic enrichment for higher-level protocol properties can be loosely decoupled with, and in general, developed relatively independently from the automation and decidability aspects of the entire proof system.

This research combines both the elegant semantic heritage of BAN-style inference systems as well as the decidability afforded by theory generation. First, a comprehensive proof system called *TGPay* is developed. *TGPay* introduces novel developments and refinements in the formal semantics that conform to the prerequisites of theory generation. Those target desired properties in payment such as confidentiality, integrity, freshness, acknowledgement and non-repudiation. Security primitives such as digital signature, message digest, message authentication code (MAC), and certificate are formally introduced in the framework. Formal semantics of message transmission and handling are also explicitly defined for senders and receivers as well as their interactions with messages.

Given a payment protocol specification, its pre-conditions and post-conditions, this thesis discusses what constitutes correctness under two scenarios. The first scenario, denoted as Alpha-S, examines the protocol under a perfect environment without any attacks. Coarsely speaking, a protocol is considered Alpha-S correct iff the corresponding pre-conditions are valid before each message exchange and all post-conditions are valid at completion.

The second scenario, known as Beta-S, deals with a second protocol, its associated pre-conditions and post-conditions, systematically modified from the original protocol. In the modified protocol, attackers are modeled explicitly as Dolev-Yao intruders [DY83] that participate in message exchanges. The modified protocol can thus be thought of as a specific attack rendered on the original protocol. Since pre-conditions and post-conditions are also obtained from this transformation, this saves protocol designers from re-defining them and hence avoids any possible errors that may arise from that process. This thesis defines what it means for a protocol to be correct under a specific Beta-S (an attack scenario).

Together with the developed formal semantics of *TGPay* and the two examination strategies Alpha-S and Beta-S, this thesis develops the Pre-conditioned Theory Generation Protocol Analyzer (PTGPA), an automatic payment protocol analyzer. PTGPA adopts theory generation as a subroutine. Leveraging the decidability property of theory generation, PTGPA fully automates analyses of candidate protocols under Alpha-S and Beta-S and is guaranteed to terminate in finite time with correctness conclusions.

This thesis examines a set of real-world payment protocols using PTGPA. The collection consists of two types of contactless credit card payment protocols and two EMV authentication protocols. Security flaws are found in some of the protocols using the formal analysis approach.

This thesis proposes two mobile NFC payment protocols and demonstrates how they are formalized and verified in the framework of PTGPA. The first protocol allows an NFC-enabled mobile phone to interact with an NFC reader to perform a general financial transaction. The second protocol allows an NFC-equipped phone with payment server connectivity to purchase services or goods that are identified by passive tags (smart cards). Unlike the design of reader-based payment protocols, the phone is now responsible for collecting transaction evidence as well as submitting it to a payment server. This thesis shows how PTGPA can be applied to analyze these protocols as well as their respective attacks in full automation.

1.2 Related Work

Although the domain is of primarily payment protocols, this thesis has benefited from studying protocol verification techniques from a much broader context. In this section, we provide a brief review of the related formal methods in the area of protocol verification, with an emphasis on theory generation and its dependent techniques.

1.2.1 Needham-Schroeder and Dolev-Yao

The history of applications of formal methods to protocol analysis goes back to the late 1970's, when [NS78] first discussed the use of encryption to achieve authentication. The paper studied examples using both symmetric key encryption as well as public key encryption techniques. Needham and Schroeder, at the conclusion of the paper, recognized the complexity of designing communication protocols for secrecy and emphasized the need for more rigorous techniques. According to [Mea03], this was perhaps the first mention of formal methods in regard to protocol analysis in the literature, although Needham and Schroeder did not address the proposal in detail.

In the early 1980's, Dolev and Yao [DY83] made the first significant attempt in applying formal mathematical models to protocol analysis. [DY83] discussed two types of restricted public-key based communication protocols: cascade protocols, in which the only operations allowed are cryptographic encryption and decryption; and name-stamp protocols, in which users are allowed to append, delete and check names encrypted with plain-text. The paper provided sufficient conditions for the secrecy properties of both types. It also suggested a polynomial runtime algorithm that is capable of deciding if a given protocol of the two types indeed ensures secrecy. The work was generalized by [EG83], in which the authors showed that checking a given protocol's security under relaxations of the constraints from [DY83] is, in general, undecidable.

Dolev and Yao's work was both promising and significant. It adopted a number of idealizations which have allowed designers to vastly simplify their formal model and concentrate more closely on the logic of a given protocol. First, cryptographic primitives are assumed to be perfect. This means that encrypted messages are unbreakable by intruders without the corresponding decryption key. Likewise, the hash functions used are collision free, and the pseudo-random generator always produces truly random numbers. The strength of the key and I/O buffer size are assumed to be adequate and are considered irrelevant in analyses. Secondly, intruders in the Dolev-Yao framework are powerful principals who can receive, modify, block, or redirect any messages transmitted in a protocol. They behave just like normal principals and are capable of initiating any communications with other regular principals. They are only limited by their knowledge of any secrets and thus any cryptographic procedures that require this knowledge. This intruder model with a fixed set of powerful capabilities has been shown to be sufficiently expressive in describing any attacks that follow the Dolev-Yao abstraction [Cer01].

Dolev-Yao abstraction quickly became the "model of models" for much subsequent work in this area. Millen et al. [MCF87] developed a Prolog program called Interrogator that searches automatically for attack vulnerability in network protocols. They used a finite state machine to represent protocol traces and to report any message modifications from attackers. In [Mea92], Meadows applied term rewriting techniques to analyze key exchange protocols. This was further extended in [Mea99], in which the well-known NRL protocol analyzer was developed and used to effectively uncover security flaws in real-world broadcast protocols.

1.2.2 State Exploration

An extension of Dolev-Yao is state exploration, where state spaces of the subject protocol are exhaustively examined. A set of transition functions is identified and formally defined. Paths that lead to desired states prove the soundness of some security claims. Paths that lead to protocol states which are successful attacks are identified and extracted as counter-examples. For instance, if one wants to prove a protocol does not reach a state in which a piece of information is revealed, one can show that it is impossible to reach that state. The proof technique can be in the form of exhaustive enumeration with prunings in which certain sets of states are proved not necessary to explore. This initially seemed to be an effective approach but was found difficult to apply, mostly due to the unbounded nature of the underlying protocol states. In particular, the message size and the number of concurrent sessions are in general unbounded. Model checkers that enumerate only finite portions of infinite state spaces do not ensure completeness. If the checker fails to find any attacks, it does not prove the protocol is attack-free.

Since then, various hybrid techniques were used to improve the applicability of the state exploration approach. For example, to overcome the problem of unboundedness, restrictions can be imposed on the message size [Ros95] [CJM00] as well as on the number of sessions [RT01] [RT03]. Notably, Lowe's early work on fixing the Needham-Schroeder public key protocol using Failures Divergences Refinement (FDR) became one of the first realizations of this hybrid technique [Low96]. Lowe's strategy was to combine manual analyses with a constrained version of state exploration which can generally be conveniently automated. This work was later extended by [Low98], in which Lowe started with an automated state exploration in constrained spaces, and showed how manual proofs can be used to aid the exploration to generalize it to the unconstrained cases. In the same vein, [EMM05] extended the NRL analyzer to incorporate a state space reduction framework with user-aided pruning. This work was subsequently extended by [EMM06].

1.2.3 Pi Calculus

A different yet popular formal verification approach is the application of equivalence theory. Equivalence theory was originally applied extensively in the Pi calculus [Mil91] [Mil99]. Pi calculus is a powerful process algebra that was primarily designed to model concurrent systems. Pi calculus was later extended by Spi-calculus [AG97] to add semantics for cryptographic primitives such as encryption and decryption. After that, Applied Pi calculus [AF01] further enriched the semantics of Spi-calculus to include additional cryptographic primitives.

In these formal frameworks, desired properties of protocols such as secrecy and authenticity are viewed as equivalent to a specification protocol. For example, message m remains confidential if a protocol exchanging message m is indistinguishable from a protocol exchanging a different message m', for any m and m'. "Indistinguishability" is claimed if one can prove the equivalence of these two protocols for any m and m'. Another property that can be proved using equivalence is authenticity. For example, a protocol that delivers message m from principal A to principal B over an insecure channel is considered to preserve authenticity if it is equivalent to a second protocol that always magically allows B to receive the same message m, for any m.

Although the framework defines a clear linkage between parameterized equivalence testing and formal proofs of secrecy and authenticity, the actual proofs are difficult due to the "for any" in the definition of equivalence, which must hold for all possible cases. Thus, practically, the proof strategy has shifted to show bisimilarities between the two protocols [San96] [AG98a] [AG98b]. Over the years, decision procedures for bisimilarity of some constrained cases were studied by a number of researchers [DSV00] [DSV03] [AC06].

1.2.4 Belief Logic

In 1989 and 1990, [BAN89] [BAN90] introduced a vastly different approach to formal verification, called BAN logic. Their framework was built on modal logic and attempts to ascertain beliefs of individual principals. This reasoning perspective is both unique and purposeful. Because principals are often exposed to different sets of messages during a protocol run, they form different sets of beliefs. Likewise, if an intruder alters a message transmission, the recipient will likely form different beliefs. BAN logic can be used to answer questions such as what beliefs principals form when a message is sent or received, or what initial assumptions were made to form these beliefs.

There were 19 rules of inference specified in the original BAN logic paper [BAN90]. These rules were developed to determine principals' beliefs regarding authentication, freshness and jurisdiction. For example, if a principal P sees a cipher-text in the form of a plain-text M encrypted using key K, and P knows the key K that P shares with Q, then P also sees plain-text M. In BAN logic, this inference rule is formalized in a decryption axiom of the form:

$$\frac{\begin{array}{c} P \triangleleft \{M\}_K \\ P \models P \overset{K}{\leftrightarrow} Q \end{array}}{P \triangleleft M} \tag{1.1}$$

where the first premise states that P sees a cipher-text that is the encryption of M using key K. The second line states that P believes it shares with Q a common key K and that P knows K. The conclusion of this axiom states that P sees M.

[Mea03] argued that belief logics such as BAN are generally weaker than state exploration due to their high abstraction level. However, the key advantage of BAN formalism is that it is very intuitive and easy to apply, yet effective in picking up loopholes in protocols. The compactness of its semantics also makes it easy to compute and to automate. Another key advantage of BAN-style formalism is that it is very flexible, since axioms can be engineered to target any specific design goal. BAN logic remains popular in the community of protocol verification.

Gaarder and Snekkenes [GS91] and van Oorchot [vO93] extended the original BAN logic to add formal semantics for public key infrastructure (PKI). This was further enhanced in [SY08], where the Gaarder and Snekkenes framework was optimized by introducing explicit handling of digital certificates.

[GNY90] suggested a set of modifications to the original BAN logic. The modifications suggested a set of new, powerful axioms. The resulting framework, called GNY logic, has been used to analyze a much wider range of protocols beyond classical authentication protocols [MO06] [DLC08] [FL09] [KP12].

The first serious attempt to verify electronic commerce protocols with BAN-style logic was done by Kailar [Kai96]. Kailar introduced formal semantics to evaluate accountability, a unique and important requirement for E-commerce transactions. Accountability dictates that participants of a protocol must be held accountable for what they have claimed and sent. Kailar points out that it is not sufficient for individual participants to believe a correct transaction took place; they must be able to prove, with evidence, to an arbitrator that this was the case. This requirement that principals are able to prove certain aspects of an E-commerce protocol is remarkable. Surprisingly, axioms of accountability can be very naturally specified in BAN-style formalism. Kailar's approach was further expanded upon by a number of researchers [SLBS08] [LHL08] [DJZF09] [SSLH11].

1.2.5 Automated Proofs

As various BAN-style formalisms were being developed, increased effort was made on the automation aspects of these frameworks. This goes back to the mid 1990's, when [KW94] first attempted to automate the BAN logic by implementing it in a logic programming language. In their paper, a modified version of the original BAN logic, called AUTLOG, was implemented in Prolog. However, the resulting implementation was sensitive to the order in which the rules were defined in Prolog, as pointed out in [Kin99]. In some cases, the order in which rules of inference were defined even led to non-termination.

Mathuria et al. [MSnN95] attempted to fully automate the GNY logic by imposing certain ad-hoc constraints on the rules of inference without altering their semantics. Mathuria et al. recognized that certain rules of inference from GNY produce conclusions that are "longer than" their premises, which do not allow a proof of the finiteness property on the number of formulas that can be derived. An example illustrated by the authors was the freshness rule:

$$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)} \tag{1.2}$$

which states that if P believes part of a message is fresh, then the whole message is fresh. Mathuria et al. pointed out that repeated application of this rule will lead to non-termination[1]. They modified it by inserting an additional premise of the same size as its conclusion. That is, this freshness rule always produces a conclusion that is no longer than what is already known. Subsequently, Mathuria et al. showed how modifications to this rule, as well as to other similar sources of non-termination, preserved the semantics of the original GNY logic and, at the same time, ensured finite-step termination. A similar attempt on the automation aspect was conducted by Monniaux [Mon99].

In parallel with BAN, various pioneering automation techniques based on finite-state exploration were developed. Specifically, Mitchell et al. [MMS97] showed how

[1] $\sharp(X) \rightarrow \sharp(X, X) \rightarrow \sharp(X, X, X)$...

general purpose state enumeration tools can be applied to security protocol verification. A general purpose state explorer, called $Mur\varphi$, was used to automatically analyze a number of finitely bounded cryptographic systems by performing an exhaustive check on all reachable states. Although Mitchell et al. did not claim any formal soundness and completeness in their paper, their work presented promising results on the automation aspect. This approach was later exploited by [SD97] for performance improvements by introducing concurrency, where multiple sessions at same time are created to increase overall analysis efficiency. Moreover, [ID96] presented an extension of the $Mur\varphi$ state explorer to verify systems with replicated identical components.

In 1999, [Kin99] [KW99] extended Mathuria et al. and Monniaux, and developed a generic framework called theory generation that is capable of performing protocol verification in full automation with any formal reasoning logic that satisfies a given set of constraints. In their work, Kindred and Wing divided inference rules into two categories: A growth-rule has a conclusion that is longer than any of its premises; a shrink-rule is one whose conclusion is shorter than some of its premises. The measurement of the size of any premise or conclusion was abstract and was rigorously defined with a customizable preorder. Given a preorder definition and with some mild syntactical constraints, Kindred and Wing showed how one can construct a *theory representation*, from a set of initial conditions, rules of inference and a given protocol description in finite number of steps. Finally, to verify if a goal formula is valid, one can simply invoke a decision procedure. This Kindred and Wing procedure was shown to be sound and complete, and was guaranteed to terminate in finite time. Case studies were examined with a general-purpose proof system called a "little logic". Theory generation and its related techniques were subsequently studied by [HSW00] and [ZF03].

Kindred and Wing's work was inspirational to this thesis. The key contribution came from its independence from any specific proof systems such as BAN, GNY or $Mur\varphi$, yet theoretical arguments were given to assert guarantees of termination and decidability. This implies that protocol designers can come up with their own set of axioms, perhaps tailored to a specific problem, without worrying about reinventing their corresponding decision procedures.

This unique feature afforded by theory generation has allowed this thesis to decompose its automated proof strategy into two parts. First, a comprehensive set of inference rules was developed to target payment. This includes rules that traditionally target secrecy and authentication as well as payment-specific rules for integrity and non-repudiation. Moreover, formal semantics were defined for message transmission and handling. The set of rules was specified according to the prerequisites of theory generation and thus the proposed proof system is amenable to Kindred and Wing's decision procedure.

Second, we introduce a set of pre-conditions and post-conditions beyond the traditional description of message transmission, and develop a decision procedure to determine the correctness of a given payment protocol with respect to these pre-conditions and post-conditions. We then formally introduce intruders in a different, modified protocol that represents an attack scenario. This second protocol is obtained mechanically from the original protocol, by applying a set of elementary capabilities from the Dolev-Yao intruder model. We develop a unified decision framework to verify the correctness of the original protocol under this specific attack, as presented by the second protocol.

1.3 Approach Overview

In this section, we give an overview of our analysis framework. We start by considering a simple NFC payment protocol example. Let Re and Ph be two principals: NFC reader and NFC phone respectively. Ph first sends a client hello message M^1_{Ph} to Re to indicate that it is ready to start the protocol. Re then sends Ph a payment request message M^1_{Re}, which contains information on the amount to charge, description of the service, etc. Upon receiving this message, Ph replies with a payment confirmation message M^2_{Ph}. M^2_{Ph} contains information on Ph's affirmation of M^1_{Re}, its credit card details, etc. Finally, after receiving M^2_{Ph}, Re sends message M^2_{Re} to Ph, where M^2_{Re} is the receipt issued to Ph. This protocol is depicted in Figure 1-1.

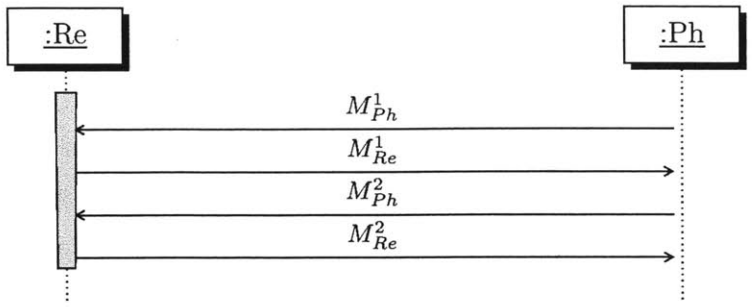


Figure 1-1: UML sequence diagram for a simple reader - NFC phone payment protocol.

This protocol has a number of potential security flaws. But for now, we focus on strategies for its correctness analyses. There are many ways one can define what constitutes a correct payment protocol. Our general strategy can be broken into two perspectives:

1. *Alpha-Scenario*: Correctness of the protocol without an intruder
2. *Beta-Scenario*: If condition 1 is met, correctness of the protocol with Dolev-Yao intruders

Throughout the proposed framework, we will formalize the messages that are exchanged, and reason with predicates in a restricted first-order logic language, to be defined in Chapter 5. We will define characteristics of the language in chapter 2, including its terms, functions and predicates. Next, we briefly describe the two examination strategies.

1.3.1 Alpha-Scenario

First, we are interested in the correctness of a protocol in a *perfect environment* without any intruders, the so-called "Alpha-Scenario". In the Alpha-Scenario (Alpha-S), we are interested in questions like: Does each message transmission achieve its expected goals? Does the protocol design comply with given security requirements at completion? We address the first question using a set of pre-conditions and the second by specifying a set of post-conditions. All pre- and post-conditions will be formalized in a specified language.

We specify a set of initial assumptions that outline the beliefs of principals prior to the start of the protocol, such as knowledge of keys. Next, for each message transmission, we define a set of pre-conditions. For example, after receiving the payment request M^1_{Re}, Ph needs to send back its payment confirmation M^2_{Ph}. However, Ph needs to first prove to itself that certain security goals are met. If Ph fails to form beliefs on some of these goals, Ph aborts the protocol. For instance, Ph needs to be able to prove that M^1_{Re} was sent recently and is not a replay of some old message, by using all the information Ph has gathered up to that point. This pre-condition can be formalized as an assertion formula:

$$Ph \models \sharp(M^1_{Re}) \tag{1.3}$$

where $Ph \models$ is a predicate that means "Ph believes" and $\sharp(M^1_{Re})$ is a predicate that means "M^1_{Re} is fresh". Notice this pre-condition does not impose a freshness requirement on a <u>specific</u> message M^1_{Re}. Rather, it should be interpreted as "For <u>any</u> message M^1_{Re} Ph receives at the end of transmission two, Ph must believe M^1_{Re} is fresh.". M^1_{Re} should be interpreted as a placeholder or a variable that represents any concrete message sent in the second transmission. In general, pre-conditions can be thought of as security and business requirements that the protocol designer imposes for each message transmission step. Alternatively during Beta-S, they can be thought of as an individual principal's vigilance against any type of attack; since failure to verify any pre-condition results in protocol abortion.

Besides pre-conditions, the proposed framework also requires specification of a set of post-conditions. Unlike pre-conditions, post-conditions are specified for the entire protocol and are not transmission specific. They can be thought of as the set of desired goals and business requirements that must be achieved at the completion of the protocol run, reasoned from each individual principal's perspective on their beliefs. One such business requirement for the above protocol is non-repudiation. For example, at the completion of the payment protocol, Ph must believe that it has non-repudiatable evidence to prove Re's recognition of his payment. If Re attempts to refute, Ph can bring forward evidence to an arbitrator to uphold its claim. Similar to 1.3, we can formalize this assertion as a formula in a formal language.

To sum up, the correctness of a given protocol in Alpha-S requires that all pre-conditions are valid before each transmission and all post-conditions are valid at protocol completion. This means that, without any intruders, a given candidate protocol must satisfy all design goals at each step while the protocol is running, and also satisfy all the design goals when the protocol successfully completes.

1.3.2 Beta-Scenario

In the Beta-Scenario (Beta-S), we are interested in knowing how a given candidate protocol that is Alpha-S correct deals with an attack. We propose a verification strategy that allows us to determine the candidate protocol's correctness under a specific attack.

To tackle this verification strategy, it is necessary to first define a structured transformation process. This transformation process is empowered through a set of Dolev-Yao intruder actions such as message eavesdropping, blockage, modification and redirection. It transforms the original protocol into a specific attack scenario, in which intruders are explicitly involved in message transmissions. The pre-conditions and post-conditions of the original protocol are also transformed in appropriate ways through the same process.

Beta-S adds descriptions of intruders' initial knowledge in the set of initial assumptions, and another set of assertions, called *Intruder Assertions*, to reason about beliefs of any Dolev-Yao intruders in an attack scenario. We are interested in what knowledge the intruders are capable of (or not capable of) constructing through interactions with honest principals in a specific way.

To illustrate the idea, we consider the following eavesdropping attack example on the above protocol as illustrated in Figure 1-2.

In this protocol, an intruder denoted I^* silently listens to all message transmissions.

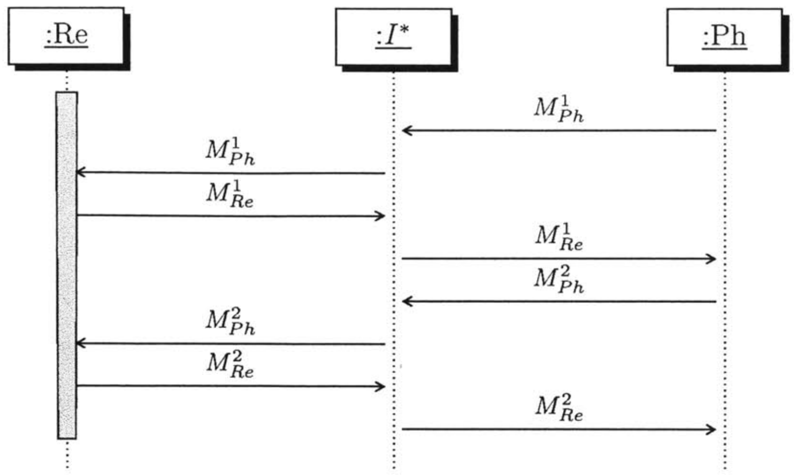


Figure 1-2: UML sequence diagram for an eavesdropping attack, where I^* is a Dolev-Yao intruder that silently listens to all message transmissions.

Ph and Re are not aware of the existence of I^*. To reason that the original protocol satisfies the secrecy requirement on the payment confirmation message M^2_{Ph}, we assert that I^* does not believe it can construct M^2_{Ph}, at any stage of the protocol run. We do this by specifying the following formula and try to formally establish its validity:

$$\neg I^* \models \sqcap\,(I^*, M^2_{Ph}) \tag{1.4}$$

where $\sqcap(I^*, M^2_{Ph})$ is a predicate that means that principal I^* can construct message M^2_{Ph} and $\neg$ is a negation operator.

To evaluate the correctness of the original protocol under any Beta-S, our framework considers two cases. First, due to the presence of intruders, certain pre-conditions in the attack protocol may be found to be invalid. In this case, the attack is said to be *detected* and the attack protocol will terminate prematurely. It follows that the original protocol is correct with respect to this attack, iff all intruder assertions are valid with respect to their knowledge set at protocol abortion.

Second, intruders may try to send modified messages to honest principals so that their pre-conditions can be met. The primary goals of intruders are not to cause a protocol to abort prematurely, but instead, to fool honest principals to believe or act on certain non-factual claims without being detected. In this case, the original protocol is correct with respect to this attack, iff for any honest principal P that has successfully finished its role without observing any abnormalities in the attack protocol as in the original protocol:

1. Each and every post-condition related to P must be valid with respect to P's entire knowledge set; and,

2. Each and every intruder assertion is valid, with respect to the entire knowledge sets of intruders.

Notice that a completed role can be either a message sender or a receiver. In the attack scenario in Figure 1-2, Re believes it has finished its role, since it has sent out its last message M^2_{Re}, and Re was satisfied with all the pre-conditions prior to sending out M^2_{Re}. Ph also believes it has finished its role, since it has received the last message M^2_{Re} as planned, given that Ph does not observe any abnormalities from M^2_{Re}. We give a rigorous definition of this intuition in Chapter 4.

In our example, assertion 1.4 will fail since M^2_{Ph} was sent in clear. I^* can construct the content of the information contained in M^2_{Ph} such as the credit card number and expiration date. Hence we conclude that the original protocol is not correct against this specific eavesdropping attack modeled in Figure 1-2. Specifically, the secrecy requirement on M^2_{Ph} was violated.

1.3.3 Proof System

Although we have discussed the high-level analysis strategy, we have not given the formal definition of what it means for a formula to be valid. To establish validity, a proof system is first developed. The proof system, written formally in a language, consists of a comprehensive set of rules of inference that target desired properties of payment such as authentication, non-repudiation, integrity, etc. Each of the inference rules has two parts: premises and conclusion. The conclusion is a single formula which can be proved valid, only if each and every premise is proved valid. For example, we can reason that if principal P received a message which is signed by the private key of Q, then P believes that Q constructed this message. This rule can be formalized as:

$$\frac{\begin{array}{c} P \models P \trianglelefteq \pi_V(X, Q) \\ P \models \wp(Q, K_P) \end{array}}{P \models \mathfrak{G}(X, Q)} \tag{1.5}$$

where the first premise says P believes that it has received a message that is some message X encrypted using Q's private key, and the second premise says that P believes that key K_P is the public key of Q. The conclusion of this inference rule says P believes that X was once uttered by Q. We emphasize that all the premises and conclusion are written from the perspective of a principal's belief, as this allows us to model different knowledge sets for different principals.

Given a query formula ϕ and a set of valid formulas Γ as a knowledge set, a proof is a sequence of applications of the rules of inference from Γ to the derivation of ϕ. ϕ is valid with respect to Γ iff there is a proof of ϕ from Γ. We adopt the Closed World Assumption (CWA). That is, all inference rules that can be invoked must be from a given proof system and the given proof system contains all the possible inference rules that can be used; no rules that are outside the proof system can be used. CWA allows us to prove as well as disprove validities of formulas.

In our framework, we will emulate each principal's reasoning about its pre-conditions by trying to prove the validities of the pre-conditions with the principal's knowledge set at the time of evaluation. The same strategy is used for post-conditions and intruder assertions. For example, in the protocol presented in Figure 1-1, the starting point is the set of initial assumptions, which are all assumed to be valid. Before Ph sends M^1_{Ph}, Ph attempts to prove all assertions of the first message's pre-conditions, using all the initial assumptions pertaining to Ph's beliefs. Ph then sends M^1_{Ph} to Re. Re will attempt to prove all assertions in the seconds message's pre-conditions, using all the initial assumptions pertaining to its beliefs plus any knowledge inferred from M^1_{Ph}, before sending out M^1_{Re}.

1.3.4 Theory Generation

Given a set of rules of inference, a knowledge base Γ and a query formula ϕ for which we wish to establish validity, it is desirable to have an efficient decision procedure to determine whether or not one can establish:

$$\Gamma \vdash \phi \tag{1.6}$$

where $\vdash$ means "is able to prove". In our framework, this role is fulfilled by a technique called theory generation.

Theory generation works by forming a finite *theory representation* $\Gamma^{\#}$, which is capable of representing a possibly infinite set of formulas that can be proved valid from Γ. [Kin99] [KW99] suggested a sound and complete procedure to determine if ϕ is valid by checking it against $\Gamma^{\#}$. The procedure is guaranteed to terminate in finite time.

Theory generation requires certain syntactical constraints when writing the set of rules of inference. Our proposed proof system, to be introduced in Chapter 5, fully satisfies these constraints and is thus amenable to the techniques developed in theory generation.

1.3.5 Framework Overview

The proposed overall analysis framework is depicted in Figure 1-3. First, a set of initial assumptions, business requirements and design goals are specified. The initial assumptions are related to each individual principal's prior knowledge and are thus shared in both Alpha and Beta scenarios. Next, we prepare a formalization of the protocol as well as its pre-conditions and post-conditions. By applying Dolev-Yao intruders, we also obtain a set of attack scenarios as well as their corresponding pre-conditions, post-conditions and intruder assertions. The specifications of the original and attack protocols are input to the PTGPA algorithm. PTGPA uses theory generation as its subroutine and checks the protocol specifications for correctness using the two examination strategies discussed above, Alpha-S and Beta-S. The theory generation step works

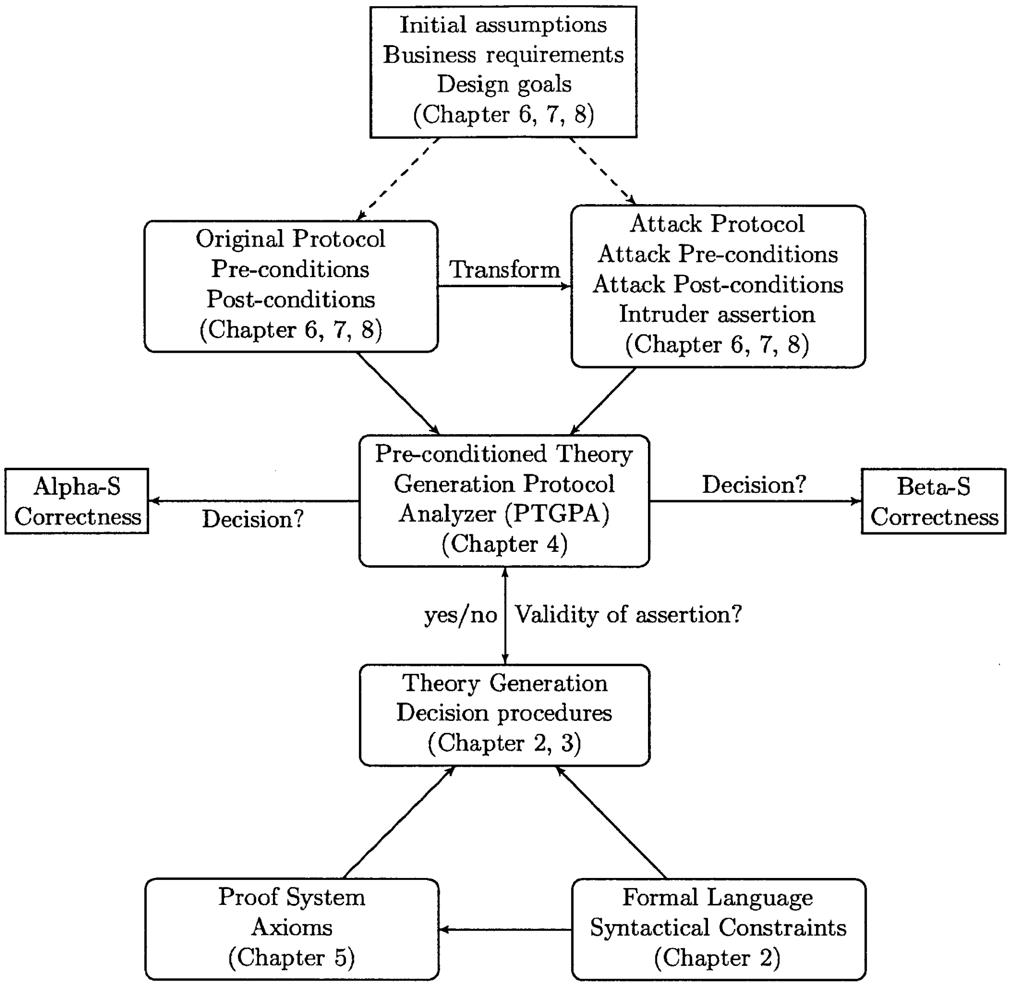


Figure 1-3: Overall analysis framework of PTGPA

with a proof system called *TGPay*, which we define for the context of payment, with respect to a set of syntactical constraints in a formal first-order logic language.

1.4 Thesis Outline

The reminder of this thesis is organized as follows: Chapter 2 provides a rigorous introduction of the formal language and theory generation, including relevant definitions and descriptions of the algorithms. Chapter 3 outlines the key properties of theory generation, including its soundness, completeness and termination. Chapter 4 presents the PTGPA analysis framework, its properties and discusses how theory generation is integrated as a subroutine. Chapter 5 presents the proposed proof system - *TGPay*, including the definitions of terms, functions and predicates. A specific preorder definition is discussed and rules of inference are specified. Chapter 6 is a case study that analyzes some real-world bank card payment protocols using PTGPA. Chapter 7 presents a case study using PTGPA to analyze a generic NFC phone - reader payment protocol, including analyses in both Alpha and Beta scenarios. Chapter 8 presents a case study using PTGPA to analyze a generic NFC phone - passive tag payment protocol, including analyses in both Alpha and Beta scenarios. The thesis concludes in chapter 9, in which we summarize this research, discuss its limitations and suggest future research topics.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Theory Generation

Contents

2.1 Overview

This chapter summarizes the technique of theory generation with a sequence of definitions and lifting lemmas. Theory generation focuses on a subset of generic first-order logic languages. The inference rules and their syntactic restrictions are presented. The main theory generation algorithm is then introduced, and this is followed by a summary of its decidability analysis in the next chapter. We maintain the original reasoning structures described in [KW97], [Kin99] and [KW99]. We add definitions and we discuss relaxations to the constraints and modifications of the original algorithm where appropriate. Any other definitions or lemmas with [Kin99] are unchanged.

2.2 Definitions

2.2.1 The L Language

Definition 1 (L language and Theory Γ^* [Kin99]). *Let L be a restricted first-order logic language that contains a finite set of rules of inference R (see Definitions 13, 14 and 15). Let Γ^0 be the set of well-formed-formulas (wffs), or just formulas, of L. The theory Γ^* (also called theory closure) generated from Γ^0 is a (possibly infinite) set of wffs that contains exactly those wffs that can be derived from Γ^0, using only R.*

Formally, we define terms and formulas in L as follows:

Definition 2 (Term in L). *Any variable is a term. Any constant is a term. Let F be a function symbol and T be an ordered set of terms. Then $F(T)$ is also a term.*

Definition 3 (Function in L). *Let T be an ordered set of terms and let F be a function symbol. Then a function in L is syntactically defined by $F(T)$ and semantically defined as a mapping from a vector of N terms to a term, $F : Term^N \rightarrow Term$.*

Definition 4 (Formula in L). *Let T be an ordered set of terms and let P be a predicate. Then a formula in L is syntactically defined by $P(T)$ and semantically defined as $P : Term^N \rightarrow \{true, false\}$.*

The syntactical structures of functions and formulas in L are similar. However, a function performs some transformations and returns the resulting term; a formula states that a certain condition holds (or does not hold) over an ordered set of terms.

Definition 5 (Substitution). *A substitution (σ) of terms for variables is a finite set of:*

$$\sigma = \{X_1 \rightarrow T_1, ..., X_n \rightarrow T_n\} \tag{2.1}$$

where each X_i is a distinct variable and each T_i is a term that is different from X_i. An instance of σ is the application of σ to a term, denoted as σT for a term T; or the application of σ to all terms of a formula P simultaneously, denoted as σP; or the application of σ to all terms of a function F simultaneously, denoted as σF.

Definition 6 (Composite Substitution). *Let θ and σ be two substitutions, and T be a term. The composition of the two substitutions, denoted as $\theta \circ \sigma$, is one such that:*

$$(\theta \circ \sigma)T = \theta(\sigma T) \tag{2.2}$$

Definition 7 (Extension of σ). *Let σ_1 and σ_2 be two substitutions and let ϕ be a formula. σ_1 is said to be an* extension *of σ_2 with respect to ϕ if for any replacement pair $\{X \to Y\}$ from σ_1 such that variable X appeared in ϕ, there is a corresponding replacement pair $X \to Y'$ in σ_2 where Y can be obtained by applying a (possibly empty) substitution to Y'.*

Following the definition, if σ_1 is an extension of σ_2 with respect to ϕ, we can write $\sigma_1\phi = (\sigma^* \circ \sigma_2)\phi$ for some substitution σ^* and some formula ϕ. Intuitively, σ_1 is an extension of σ_2, if σ_1 is more specific than σ_2, or is "subsumed" by σ_2. In our framework, extensions are only defined with respect to formulas, but not for functions.

Definition 8 (Unification). *Given a set of formulas, a unifier is a substitution that makes the formulas of the set identical.*

With the definitions of formulas above, $P(X,Y)$ and $P(Z,X)$ are both formulas in L. Although they are symbolically different, they mean essentially the same thing. We will eliminate repetitions of this symbolic type by using canonical representation, in which we define equivalence of formulas using modulo variable renaming. Formally, variable renaming is defined as:

Definition 9 (Modulo Variable Renaming [Kin99]). *Two formulas ϕ_1 and ϕ_2 are equivalent modulo variable renaming iff there exists a substitution σ such that,*

$$\phi_1 \overset{symbolic}{=} \sigma\phi_2 \tag{2.3}$$

where σ replaces variables from ϕ_2 with other variables.

We use $\phi_1 \overset{symbolic}{=} \phi_2$ when the two formulas are symbolically equivalent. We will use the "=" operator in the rest of this chapter and assume it means symbolically equivalent.

Definition 10 (Grounded Term/Formula [Kin99]). *A term is said to be grounded iff it does not contain any variables. A formula ϕ is said to be grounded iff all terms within ϕ are grounded.*

The theory generation algorithm, which we will introduce momentarily, requires that all initial assumptions be grounded. They must be formed only from predicates of constant arguments such as keys and principal names, or in general, message strings that do not contain any variables.

2.2.2 Rules of Inference

Next, the rules of inference (axioms) used in L are introduced. The rules are partitioned into three categories: Shrink-Rule, Growth-Rule and Rewrite. Each definition is accompanied by its syntactical restrictions. Those restrictions are important in the termination analysis of the theory generation algorithm.

We define an abstract order relation between two formulas or two terms called $\preceq$. Roughly speaking, $\preceq$ is an order on the sizes of two formulas or terms.

Definition 11 (Preorder $\preceq$ [Kin99]). *Let ϕ, ϕ_1, ϕ_2 be formulas in L; T, T_1, T_2 be terms in L; and let X be a variable defined in L. A preorder (both reflexive and transitive) relationship $\preceq$ is one that satisfies all the following:*

1. *Monotonicity: If term T_1 is <u>shorter</u> than T_2, then substituting free variable X within formula ϕ using T_1 will result in a <u>shorter</u> formula than substituting X with T_2. Formally, let $\sigma_1 = \{X \rightarrow T_1\}$ and $\sigma_2 = \{X \rightarrow T_2\}$, and let $\models$ be the "logically imply" or "result in" operator. It then holds that:*

$$(T_1 \preceq T_2) \models (\sigma_1\phi \preceq \sigma_2\phi) \tag{2.4}$$

2. *Substitution Preservation: Let T be a term, ϕ_1 and ϕ_2 be two formulas such that $\phi_1 \preceq \phi_2$ and let X be a variable from either ϕ_1 or ϕ_2 or both. Then, substituting X with T does not affect the order between ϕ_1 and ϕ_2 with respect to $\preceq$. Formally, let $\sigma = \{X \rightarrow T\}$. It then holds that:*

$$(\phi_1 \preceq \phi_2) \models (\sigma\phi_1 \preceq \sigma\phi_2) \tag{2.5}$$

3. *Finite Modulo Variable Renaming: The set $\{\phi | \phi \preceq \phi^*\}$ must be* finite *modulo variable renaming for any formulas ϕ^* in language L.*

The third condition is a key condition that is required for algorithm termination. That is, given a formula ϕ^*, we require that formulas that are *shorter* than ϕ^* be finite in number.

Definition 12 (Rule of Inference). *The rule of inference used in L is* Modus Ponens. *Let $P = \{P_1, ..., P_n\}$ be a set of n formulas denoting premises and C be a* single *formula denoting a logical consequence. A rule of inference R, in language L is of the form:*

$$\frac{P_1, ..., P_n}{C} \tag{2.6}$$

This definition states that, if the premises P are all satisfied, then the logical consequence C can be derived. Alternatively, from a principal's point of view, if $P_1, ..., P_n$ are known, then C is also known.

We will subsequently use R to denote a single rule or a set of inference rules. All the inference rules defined in L are in this form. However, there exist differences among them. In the next set of definitions, we introduce the different types of inference rules in theory generation.

Definition 13 (Shrink-Rule (S-Rule) [Kin99]). *An S-Rule S is an inference rule in which some of its premises are designated as* primary premises. *Let P_i be any primary* *premise of S and C be the conclusion of S; it is required that:*

$$C \preceq P_i \tag{2.7}$$

That is, we require that the conclusion be shorter than <u>any</u> primary premise. An S-Rule of inference must have at least one primary premise.

Definition 14 (Growth-Rule (G-Rule) [Kin99]). *A G-Rule G is an inference rule such that <u>each and every</u> of its premises P_i satisfies:*

$$P_i \preceq C \tag{2.8}$$

where C is the conclusion of G.

Definition 15 (Rewrite Rule (W-Rule) [Kin99]). *A W-Rule W is one that transforms ϕ_1 to ϕ_2 by swapping the order of the terms of functions or predicates. A W-Rule must satisfy:*

$$\phi_1 \preceq \phi_2, \qquad \phi_2 \preceq \phi_1 \tag{2.9}$$

and hence W-Rule invocations are always size-preserving.

A W-Rule expresses that ϕ_1 can be replaced with ϕ_2, and in general, vice versa. Additionally, a W-Rule does not change the size of ϕ_1 or ϕ_2 with respect to $\preceq$.

2.2.3 Proof and Validity in L

In this subsection, we formally introduce the notion of proof and validity using rules of inference. We will also introduce the Negation as Failure (NAF) assumption and the definition of disproof.

Definition 16 (Proof $\mathfrak{P}$ [Kin99]). *A proof $\mathfrak{P}$ of some grounded formula ϕ^* from a set of grounded formulas Γ is a finite sequence $\{\phi_1, \phi_2 ..., \phi^*\}$ of grounded formulas, where* *for each ϕ_i in the sequence, either $\phi_i \in \Gamma$, or ϕ_i is the result of an application of one of the rules of inference using $\{\phi_1, ..., \phi_{i-1}\}$.*

With the definition of proofs in L, we now formally introduce the establishment of validity and invalidity of formulas.

Definition 17 (Validity). *A grounded formula ϕ^* is valid with respect to a set of grounded formulas Γ and a set of rules of inferences R, if there exists a proof $\mathfrak{P}$ of ϕ^* from Γ using R.*

Next, we introduce the provability $\vdash$ operator. While $\models$ means, at high level of abstraction, "logically imply" or "result in", $\vdash$ has a precise definition.

Definition 18 (Validity Operator $\vdash$). *Under a set of rules of inference R, we write $\Gamma \vdash \phi^*$ for a grounded formula ϕ^*, iff there is a proof of ϕ^* from Γ using R. We write $\Gamma \vdash^{\mathfrak{P}} \phi^*$ iff $\mathfrak{P}$ is a proof of ϕ^* from Γ using R. In both cases, ϕ^* is said to be derivable from Γ using R.*

Definition 19 (Invalidity Operator $\nvdash$). *Under a set of rules of inference R, a set of formulas Γ and a goal formula ϕ^* that is grounded, we write $\Gamma \nvdash \phi^*$ iff there does* not *exist a proof of ϕ^* from Γ using R.*

A central assumption we will make for our proof system is Negation As Failure (NAF). Roughly speaking, if ϕ^* cannot be derived, then formula ϕ^* is not valid. Formally, this is:

Definition 20 (NAF as Invalidity). *Given a set of rules of inference R, a set of grounded formulas Γ and a grounded goal formula ϕ^*, iff*

$$\Gamma \nvdash \phi^* \tag{2.10}$$

Then ϕ^* is invalid.

Since ϕ^* is grounded, it can be thought of as an assertion of a fact. The proof system answers "yes" if there is a proof of ϕ^* from Γ and R, and the proof system answers "no" if there does not exist a proof of ϕ^* from Γ and R.

Definition 21 (Negation Operator $\neg$). *$\neg\phi^*$ is valid iff ϕ^* is invalid.*

By the above definition, to prove $\neg\phi^*$, one only needs to show that ϕ^* is invalid using NAF. One can disprove $\neg\phi^*$ if one is able to prove the validity of ϕ^*.

2.3 Finite Representation

Coming back to the first definition of the chapter, we want to generate and represent the theory *closure* Γ^* (possibly infinite) from an initial set of formulas Γ^0 with the help of inference rules from R. If Γ^* can be generated, the validity of any query formula can be determined using a simple membership function. The technical difficulty is that Γ^* is possibly infinite and therefore the generation process never terminates. To avoid this problem, a *theory representation* technique is developed by [Kin99]. We present an example of a deductive process to illustrate the issue and to solidify the definitions above, before introducing the definition of *theory representation.*

2.3.1 An Example

Suppose a communication channel is initialized between principals "Alice" and "Bob". Let $\{MSG\}_K$ be an AES [NF02] encryption function that encrypts the payload MSG with key K. Let $\theta(MSG)$ be a SHA-1 [IETF12c] hash function that generates message MSG's SHA-1 digest. Define predicate symbol $See(Person, Message)$ to be that a Person sees a particular Message and, $Know(Person, Something)$ to be that a Person knows Something. Let predicate $Link(Channel, Person1, Person2)$ be that a Channel is established between Person1 and Person2. Define $Send(Person, MSG, Channel)$ to mean that a Person sent MSG over Channel.

Rules of Inference

The set of rules of inference R contains 2 S-Rules, 1 G-Rule and 1 W-Rule:

S-Rule (Message Receiving). *If Person1 has sent MSG over Channel, and Channel is* *shared between Person1 and Person2, then Person2 sees MSG.*

$$\frac{Send(Person1, MSG, Channel), Link(Channel, Person1, Person2)}{See(Person2, MSG)} \tag{2.11}$$

S-Rule (Decryption). *If Person sees MSG encrypted using key K and Person knows K, then Person sees MSG.*

$$\frac{See(Person, \{MSG\}_K), Know(Person, K)}{See(Person, MSG)} \tag{2.12}$$

G-Rule (SHA-1 Hash). *If Person sees MSG, then Person can compute the hash of MSG and therefore can see the hash of MSG.*

$$\frac{See(Person, MSG)}{See(Person, \theta(MSG))} \tag{2.13}$$

W-Rule (Link Associativity). *This W-Rule expresses associativity of channel between two principals. If Channel is between Person1 and Person2, it is also between Person2 and Person1.*

$$\frac{Link(Channel, Person1, Person2)}{Link(Channel, Person2, Person1)} \tag{2.14}$$

The initial set of valid formulas Γ^0 before the protocol starts is the initial set of assumptions $\Gamma^{Assumption}$:

$$\Gamma^{Assumption} = \{Know(Alice, k), Link(nfc, Alice, Bob)\} \tag{2.15}$$

as well as the single communication protocol message $\Gamma^{Protocol}$ subject to analysis:

$$\Gamma^{Protocol} = \{Send(Bob, \{m\}_k, nfc)\} \tag{2.16}$$

$$\Gamma^0 = \Gamma^{Protocol} \cup \Gamma^{Assumption} \tag{2.17}$$

where m is a specific message and k is a specific key and nfc is a specific channel. The goal formula ϕ^* that we wish to query for validity, after this simple protocol ends, is that Alice can compute the hash of the hash of m:

$$\phi^* = See(Alice, \theta(\theta(m)) \tag{2.18}$$

Example Deduction

We now attempt to validate the goal by manually applying the rules of inference. We use Γ^i to keep track of formulas that are proved valid up to iteration i. Invoking W-Rule 2.14, we derive $Link(nfc, Bob, Alice)$. Therefore:

$$\Gamma^1 = \Gamma^0 \cup \{Link(nfc, Bob, Alice)\} \tag{2.19}$$

Next we invoke S-Rule 2.11, and we derive $See(Alice, \{m\}_k)$ from 2.19. We have:

$$\Gamma^2 = \Gamma^1 \cup \{See(Alice, \{m\}_k)\} \tag{2.20}$$

Invoking S-Rule 2.12 from 2.20, we have:

$$\Gamma^3 = \Gamma^2 \cup \{See(Alice, m)\} \tag{2.21}$$

Invoking G-Rule 2.13 from 2.21, we have:

$$\Gamma^4 = \Gamma^3 \cup \{See(Alice, \theta(m))\} \tag{2.22}$$

Invoking G-Rule 2.13 again on 2.22, we have:

$$\Gamma^5 = \Gamma^4 \cup \{See(Alice, \theta(\theta(m)))\} \tag{2.23}$$

We find that $\phi^* \in \Gamma^5$ and therefore we have proved ϕ^* in 5 iterations using the rules of inference in R. A corresponding proof $\mathfrak{P}$ is shown graphically on the left side

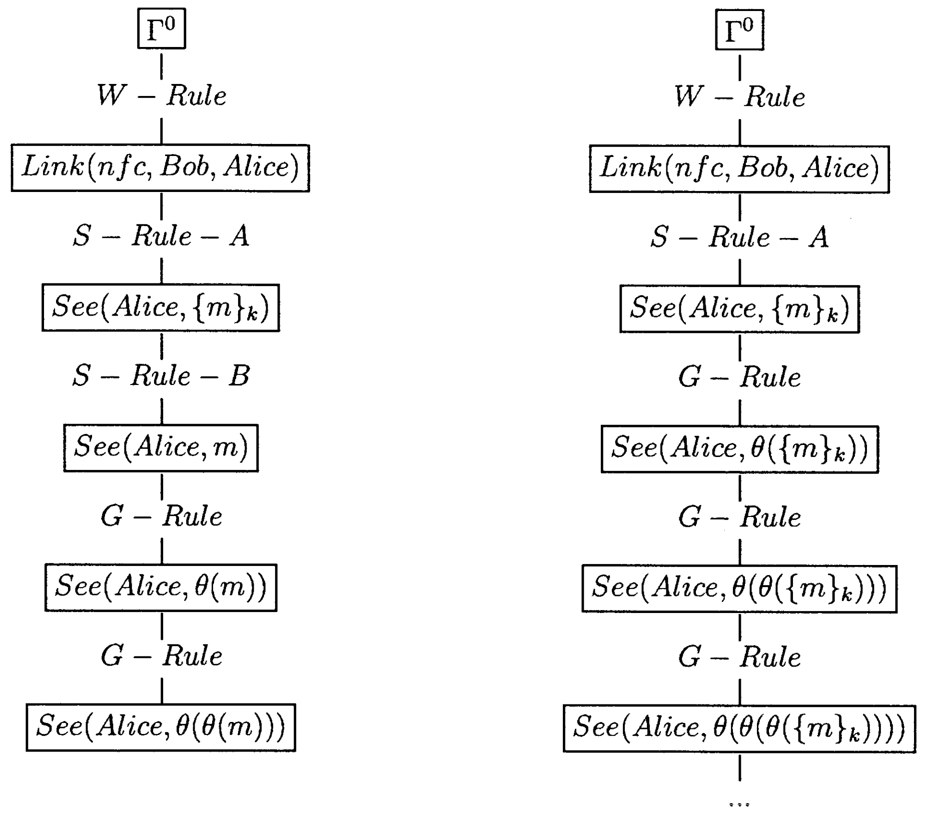


Figure 2-1: Left: A proof trace of ϕ^* from Γ^0 using R. Right: An attempt to ϕ^* from Γ^0 using R with preference of G-Rule application with infinite depth.

of Figure 2-1, where newly derived formulas are boxed. Rules of inference applied at each iteration are labeled along the edges.

In this example, we have been manually applying rules of inference at each iteration. For more complex problems, a search algorithm that applies rules of inference in a structured way needs to be developed. The choice of rule to apply at each iteration is important. Different choices will result in different proofs and sometimes lead to traces with infinite depth, even though a valid proof clearly exists. For example, the right side of figure 2-1 shows an incomplete proof of the same target formula ϕ^* with the same Γ^0 and R, but with infinite depth and therefore never terminating.

The direct consequence of non-termination is the loss of provability (or loss of disprovability), since one can only conclude if ϕ^* is valid or not valid after a search completes.

Theory generation eliminates the problem of non-termination by generating a finite *theory representation* using a structured search algorithm. At each iteration, the search algorithm performs a local search using an S-Rule that is guaranteed to return in finite time. Since there is a finite number of formulas to search for in the *theory representation*, the entire representation generation procedure can be shown to always terminate in finite time. Next, we formally define *theory representation* and introduce the theory generation algorithm.

2.3.2 Theory Representation

Definition 22 ({R,R'} Representation [Kin99]). *Let R be a set of inference rules, $R' \subseteq R$. An (R, R') representation of a theory induced by* Γ^0 *is a set that contains* all *the formulas that can be derived from* Γ^0 *using R, among which, for each such formula, the* last inference rule *used to derive it is a member from R'. Moreover, all formulas contained in this set are of this form.*

Specifically, theory generation requires that $R = \{Srules \cup Grules \cup Wrules\}$ and that only the S-Rules are used as $R's$ in the (R, R') representation of the theory. In this way, [Kin99] showed that the theory representation is finite and the algorithm generating it terminates in finite time. This is a necessary condition for the proof of its decidability property.

2.4 Prerequisites

Before describing the main theory generation algorithm, additional syntactical constraints are imposed to ensure algorithm termination. This section presents these syntactical constraints and summarizes the set of prerequisites for theory generation.

The syntactical restriction, Definition 23, is on writing S-Rules and it controls the interactions between S-Rules and G-Rules.

Definition 23 (S/G Restriction - [Kin99]). *Given an S-Rule with any arbitrary primary premise P_i and any arbitrary non-primary premise S_j and a conclusion C, S/G restriction requires that:*

1. *P_i does not unify with conclusions of any G-Rules, and*

2. *For any such S_j, it holds that $(S_j \preceq C)$*

Definition 23 can be relaxed, resulting in Definition 24. We will show in the next chapter that this relaxation leads to the same conclusion of the termination of theory generation.

Definition 24 (S/G Restriction - New). *Given an S-Rule with any arbitrary primary premise P_i and any arbitrary non-primary premise S_j, S/G restriction requires that:*

1. *P_i does not unify with conclusions of any G-Rules, and*

2. *For any combination of S_j and P_i, it holds that $(S_j \preceq P_i)$*

Putting everything together, the prerequisites of the theory generation are as follows:

Definition 25 (Prerequisites for Theory Generation [Kin99]). *The theory generation algorithm requires prerequisites to hold over a finite set of initial assumptions Γ^0, a finite set of rules of inference (R), as well as a syntactical constraint (S/G Restriction), and a computable preorder definition $\preceq$:*

1. *Operator $\preceq$ satisfies Definition 11;*

2. *Every formula in the initial assumption set Γ^0 is grounded;*

3. *Every rule in R is either a G-Rule or an S-Rule or a W-Rule, with respect to operator $\preceq$. One such partition of all rules in R must exist;*

4. *Every W-Rule rule W is size-preserving, with respect to operator $\preceq$. For any premise P and conclusion C in rule W:*

$$P \preceq C \tag{2.24}$$

$$C \preceq P \tag{2.25}$$

5. *S/G restriction holds for all S-Rules, with respect to operator $\preceq$.*

2.5 Algorithms

This section outlines the theory generation and formula derivation algorithms. The first part presents a set of algorithms that perform theory representation generation. The second part presents an algorithm that determines if a candidate formula is valid, using the theory representation output from the first part. The soundness and completeness properties of the algorithms are studied in Chapter 3.

2.5.1 *theory_gen*() Algorithm

The *theory_gen*() algorithm produces an $\{R, S\_Rules\}$ representation $\Gamma^{\#}$ of the theory Γ^* mechanically, using inference rules $R = \{G\_Rules, S\_Rules, W\_Rules\}$, a definition of $\preceq$, and initially valid formulas Γ^0. It first checks to see if the prerequisites are met. This algorithm is given in Figure 2-2.

The *closure*() function in Algorithm 2 takes *fringe*, a set of newly proved formulas from the previous call to itself, and a set of currently known formulas Γ, and produces the $\{R, S\_Rules\}$ representation of the theory Γ^*. At each step it tries to apply any S-Rules possible from the the set of all known formulas, denoted as Γ'. These newly derived formulas form the *fringe* which becomes the first argument to the next iteration of the *closure*() call. The details of the recursive function are specified in Figure 2-3.

A schematic illustration of the *closure*() function is presented at Figure 2-4.

```
Algorithm 1: theory_gen ($\Gamma^0, R, \preceq$)
  Data: $\Gamma^0, R, \preceq$
  Result: Generate theory representation
  if $prerequisites\_violate(\Gamma^0, R, \preceq)$ then
      return BAD;
  else
      $\Gamma^{\#} = closure(\Gamma^0, \emptyset)$;
      return $\Gamma^{\#}$;
  end
```

Figure 2-2: Theory generation algorithm: *theory_gen*() function [Kin99].

```
Algorithm 2: $closure(fringe, \Gamma)$
  Data: Newly proved formulas ($fringe$), knowns $\Gamma$
  Result: Generate theory representation from $\Gamma \cup fringe$
  if $fringe = \emptyset$ then
      return $\Gamma$;
  else
      $\Gamma' \leftarrow \Gamma \cup fringe$;
      $fringe' \leftarrow \bigcup_{S \in S\_Rules} apply\_S\_Rule(S, \Gamma') \backslash \Gamma'$;
      return $closure(fringe', \Gamma')$;
  end
```

Figure 2-3: Theory generation algorithm: *closure*() function [Kin99].

At the top left of Figure 2-4, $\Gamma' = \Gamma^0$ when entering *closure*() for the first time. Top right: Three new formulas are proved valid from applications of S-Rules. The three formulas form $fringe'$. $closure(fringe', \Gamma')$ is invoked. Bottom left: In the second iteration to *closure*(), the new Γ' is formed by adding the three new formulas to all the knowns from the top left. Bottom right: Another two formulas are proved valid.

The *apply_S_Rule*() function, Algorithm 3, as illustrated in Figure 2-5, takes a specific S-Rule S and the set of all formulas derived so far Γ, and tries to satisfy the premises of S by calling the *backward_chain*() function. The *backward_chain*() function returns a set of possible substitutions under which the conclusion of the S-Rule is valid.

The *backward_chain*() function, Algorithm 4, as illustrated in Figure 2-6, takes a set of formulas from an S-Rule's premises and tries to satisfy them using G-Rules

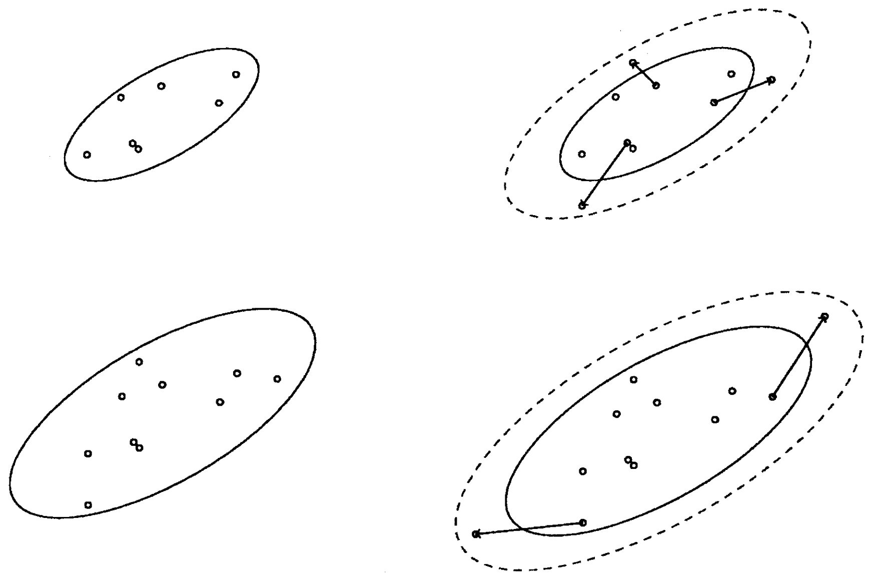


Figure 2-4: A schematic illustration of the operation of *closure*() function.

```
Algorithm 3: $apply\_S\_Rule(S, \Gamma)$
    Data: S-Rule S, knowns $\Gamma$
    Result: Apply S in every possible way, using only $G\_Rules$ and $W\_Rules$
    Sub $\sigma = backward\_chain(Premises(S), \Gamma)$;
    return $\sigma Conclusion(S)$;
```

Figure 2-5: Theory generation algorithm: *apply_S_Rule*() function [Kin99].

and W-Rules only. It tries to satisfy all the primary premises first before moving on to any non-primary premises. It tries to instantiate each formula by calling the *backward_chain_one*() function. After a possible substitution is returned for this formula, *backward_chain*() applies the substitution to all the remaining formulas and then recursively calls itself with the the remaining formulas as arguments.

The *backward_chain_one*() function, Algorithm 5, as illustrated in Figure 2-7, takes a single formula and tries to satisfy it using G-Rule and W-Rules only. It keeps track of all visited formulas and ensures that only unique formulas are considered. For any given formula, it will try to instantiate directly with any grounded formulas ϕ in the known set

```
Algorithm 4: $backward\_chain(goals, \Gamma)$
  Data: Premises to instantiate (goals), knowns $\Gamma$
  Result: Return all substitutions under each of which the goals are
          simultaneously valid, using G-Rules and W-Rules only from $\Gamma$
  if $goals = \emptyset$ then
    return $\emptyset$;
  else
    $\{first\_goal, other\_goals\} = goals$;
    $\sigma_1 \in backward\_chain\_one(first\_goal, \Gamma)$;
    $\sigma_2 \in backward\_chain(\sigma_1 other\_goals, \Gamma)$;
    return $\sigma_2 \circ \sigma_1$;
  end
```

Figure 2-6: Theory generation algorithm: *backward_chain*() function [Kin99].

Γ. Additionally, it also seeks to instantiate this formula by calling *reverse_apply_grule*() for some G-Rule. Possible substitutions resulting from both methods are returned.

```
Algorithm 5: $backward\_chain\_one(g, \Gamma)$
  Data: A single premise g to instantiate, knowns $\Gamma$
  Result: Return all substitutions under each of which g is valid, using G-Rules
          and W-Rules from current knowns $\Gamma$
  $direct\_subsets \leftarrow \bigcup_{\phi \in \Gamma} unify(g, \phi)$;
  $grule\_subsets \leftarrow \bigcup_{G \in G\_Rules} reverse\_apply\_grule(G, g, \Gamma)$;
  return $\{direct\_subsets \cup grule\_subsets\}$;
```

Figure 2-7: Theory generation algorithm: *backward_chain_one*() function [Kin99].

```
Algorithm 6: $reverse\_apply\_grule(G, g, \Gamma)$
  Data: A single premise g to instantiate, a given G-Rule G, knowns $\Gamma$
  Result: Return all substitutions under each of which g is valid from $\Gamma$, using
          proofs whose last rule of application is G
  $\sigma_3 \in unify(g, Conclusion(G))$;
  $\sigma_4 \in backward\_chain(\sigma_3 premises(G), \Gamma)$;
  return $\{\sigma_4 \circ \sigma_3\}$;
```

Figure 2-8: Theory generation algorithm: *reverse_apply_grule*() function [Kin99].

The *reverse_apply_grule*() function, Algorithm 6 takes a single formula and a single

G-Rule whose conclusion matches the formula. It tries to satisfy the premises of the G-Rule by calling the *backward_chain*() function. It return all substitutions such that under each of which g is valid from Γ, using proofs whose last rule application is the G-Rule G.

2.5.2 *derivable*() Algorithm

After theory representation $\Gamma^{\#}$ is produced by the *theory_gen*() function in Algorithm 1, one invokes the *derivable*() function in Algorithm 7 to determine if a grounded formula ϕ^* is a logical consequence (valid) of the initial assumptions Γ^0. The *derivable*() function returns a boolean that indicates if the candidate formula is valid with respect to the set of initial assumptions, the set of rules and the specified preorder. It invokes *backward_chain*($\{\phi^*\}, \Gamma^{\#}$) to see if an empty set is returned.

```
Algorithm 7: derivable($\phi^*, \Gamma^{\#}$)
  return backward_chain($\{\phi^*\}, \Gamma^{\#}) \overset{?}{=} \emptyset$;
```

Figure 2-9: Derivation algorithm: *derivable*() function [Kin99].

2.6 Summary

This chapter rigorously introduced theory generation, a general purpose framework for determination of validity of a grounded query formula. Theory generation works with a formal language L and some syntactical constraints. Instead of performing a membership check directly with Γ^* - the theory closure that is possibly infinite, theory generation first constructs a finite theory representation $\Gamma^{\#}$. It then invokes a search procedure *derivable*(ϕ^*, $\Gamma^{\#}$) to determine if query formula ϕ^* is valid with respect to Γ^0 and R.

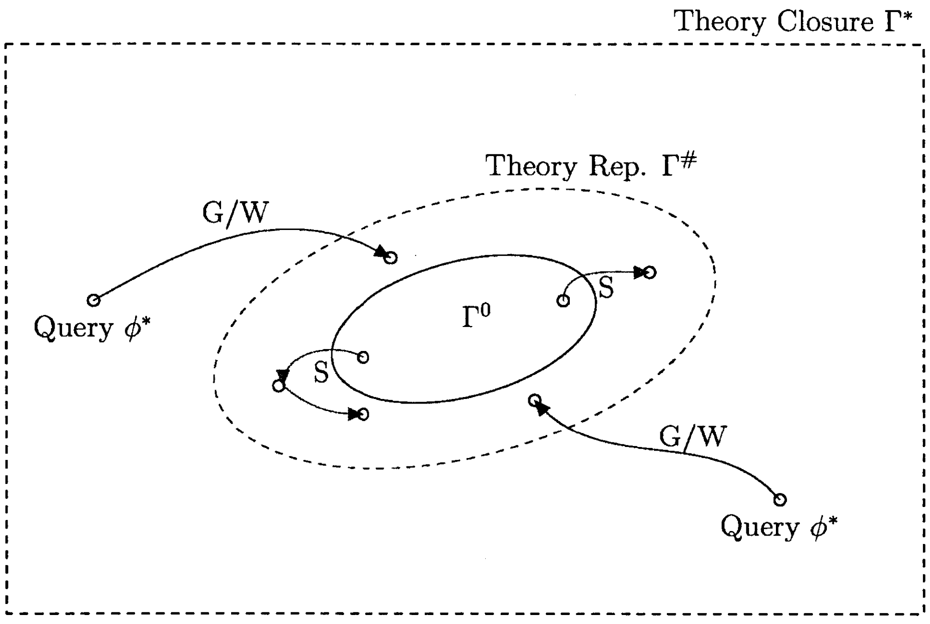


Figure 2-10: An illustration of the theory generation procedures (Derived from [KW99]).

An illustration of the entire procedure is presented in Figure 2-10. We start with Γ^0, the set of initial assumptions (the solid ellipse). We then repeatedly apply S-Rules to discover new formulas. We will eventually reach a fixed point, the dashed ellipse representing the finite theory representation. To decide the validity of a query formula ϕ^*, we invoke a procedure that searches from the query formula backwards to see if it is possible to land somewhere within the theory representation, by using G-Rules and W-Rules only.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Termination and Decidability

Contents

3.1 Introduction

In the previous chapter, the *theory_gen*() and *derivable*() theory generation algorithms were presented. In this chapter, we discuss some important properties of these algorithms. These properties are the soundness, completeness and termination of theory generation.

Loosely speaking, soundness requires that any grounded formulas returned by theory generation are correct, in the sense that there exists a corresponding proof from the initial assumptions by using the defined rules of inference and that the last rule of inference used is an S-Rule. The completeness result requires that, if there exists a proof of formula ϕ_1 from the initial assumptions by using the defined rules of inference and the last rule of inference used is an S-Rule, then there is a corresponding formula ϕ_2 returned by the theory generation algorithm such that $\phi_2 \vdash \phi_1$. Finally, the termination result provides theoretical argument on the algorithms' termination in finite number of steps. That is, theory generation always returns a result in finite time.

This section starts by presenting the relevant lifting lemmas. Some of these lemmas appeared in [Kin99] without proofs. We have re-defined them to clarify [Kin99]. These lemmas are followed by the theorems on soundness, completeness and algorithm termination. All proofs are given in Appendix A. We make reference to [Kin99] whenever the proofs are the same or of similar structure.

3.2 Lifting Lemmas

Definition 26 (Proofs with G-Rule and W-Rule). *We define two forms of proof in L. Let Γ be a set of formulas. We write $\Gamma \vdash_W \phi^*$ if there exists a proof of ϕ^* from Γ using only W-Rules and instantiation. We write $\Gamma \vdash_{GW} \phi^*$ if there exists a proof of ϕ from Γ using only G-Rules, W-Rules and instantiations (but no S-Rules).*

We next present the lifting lemmas.

Lemma 1 (Rewrite Substitution). *If ϕ_1 and ϕ_2 are formulas of L, and σ is a substitution such that*

$$\sigma\phi_1 \vdash_W \sigma\phi_2 \tag{3.1}$$

then,

$$\phi_1 \vdash_W \sigma\phi_2 \tag{3.2}$$

Lemma 2 (Unify soundness [Kin99]). *If ϕ_1 and ϕ_2 are formulas of L, and σ is a unifier* *of ϕ_1 and ϕ_2 in Definition 8, then:*

$$\phi_1 \vdash_W \sigma\phi_2 \tag{3.3}$$

$$\phi_2 \vdash_W \sigma\phi_1 \tag{3.4}$$

Lemma 3 (Constraint Unify completeness). *Let ϕ_1 and ϕ_2 be two formulas. Let ϕ_1 be grounded and let σ be a substitution, such that*

$$\phi_1 \vdash_W \sigma\phi_2 \tag{3.5}$$

then for any substitution σ' such that σ' is a unifier of ϕ_1 and ϕ_2 in Definition 8, σ is an extension of σ' with respect to ϕ_2.

3.3 Soundness and Completeness

In this section, we present the soundness and completeness claims on the theory generation algorithms. The theorems originally appeared in [Kin99]; here we will present the set of claims only.

3.3.1 Soundness

Theorem 1 (*backward_chain*() Soundness [Kin99]). *Let Φ be a set of formulas in L, Γ be a set of grounded formulas in L, and let σ be a substitution such that:*

$$\sigma \in \mathit{backward\_chain}(\Phi, \Gamma) \tag{3.6}$$

Then there exists a proof of $\sigma\phi$ for each and every $\phi \in \Phi$, from Γ using only instantiations, G-Rules or W-Rules.

Theorem 2 (*closure*() Soundness [Kin99]). *Let Γ and $fringe$ be two sets of grounded formulas in L; let ϕ be a formula such that $\phi \in closure(fringe, \Gamma)$. Then there exists* *a proof $\mathfrak{P}$ of ϕ whose last rule of application (if any) is an S-Rule:*

$$(\Gamma \cup fringe) \vdash \phi \tag{3.7}$$

Theorem 2 states that any formula ϕ generated from the *closure*() function is valid in the theory representation. We next claim the reverse, that any formula that is valid in theory representation is also generated by the closure function.

3.3.2 Completeness

Theorem 3 (*backward_chain*() Completeness [Kin99]). *Let Γ be a set of grounded formulas in L, $\Phi = \{\phi_1, ..., \phi_n\}$ and σ' be a substitution. Let $\mathfrak{P}^* = \{\mathfrak{P}_1, ..., \mathfrak{P}_n\}$ be the set of corresponding proofs of grounded term $\sigma'\phi_i$ from Γ using only instantiation, G-Rules or W-Rules. Then for any substitution σ such that:*

$$\sigma \in backward\_chain(\Phi, \Gamma) \tag{3.8}$$

σ' is an extension of σ with respect to any $\phi_i \in \Phi$.

Definition 27 (Partial Theory Representation [Kin99]). *Let Γ and $fringe$ be two sets of grounded formulas. If, for any formula, ϕ, and a proof $\mathfrak{P}$:*

1. *$\Gamma \vdash^{\mathfrak{P}} \phi$, and,*

2. *$\mathfrak{P}$ contains only one S-Rule application (and possibly many G-Rule and W-Rule applications), and,*

3. *That S-Rule application is the last rule applied in $\mathfrak{P}$, and,*

4. *$(fringe \cup \Gamma) \vdash_W \phi$,*

then we call the ordered pair $\{fringe, \Gamma\}$ a partial theory representation.

Intuitively, fringe is the set of *all* valid formulas that can be immediately proved from Γ, using a single S-Rule (and optionally using some G or W-Rules before that single S-Rule).

Theorem 4 (*closure*() Completeness [Kin99]). *Let $\{fringe, \Gamma\}$ be a partial theory representation in Definition 27. Then for any formula ϕ and proof $\mathfrak{P}$ (if any), whose last rule of application is an S-Rule application, where*

$$fringe \cup \Gamma \vdash^{\mathfrak{P}} \phi \tag{3.9}$$

there exists a formula ϕ' where

$$\phi' \in closure(fringe, \Gamma) \tag{3.10}$$

such that $\phi' \vdash_W \phi$

3.4 Termination Analysis

In this section, we present the claims for the theory generation algorithms' termination. The termination proof is constructed over a number of lemmas. We start by giving a definition on size-boundedness.

Definition 28 (Size-boundedness [Kin99]). *A formula ϕ is size-bounded by a finite set of formulas Γ, when for any substitution σ, there exists $\phi^* \in \Gamma$ such that:*

$$\sigma\phi \preceq \phi^* \tag{3.11}$$

By the above definition, if ϕ is grounded and Γ is a set of grounded formulas, then ϕ is said to be bounded by Γ if there exists ϕ^* from Γ such that $\phi \preceq \phi^*$.

Lemma 4. *If ϕ_1 is size-bounded by Γ, and $\phi_2 \preceq \phi_1$, then ϕ_2 is size-bounded by Γ. [Kin99]*

Lemma 5. *If ϕ_1 is size-bounded by Γ, then for any ϕ_2 such that $\phi_1 \vdash_W \phi_2$, ϕ_2 is also size-bounded by Γ. [Kin99]*

Lemma 6. *If Γ is a finite set of formulas, then there are finite numbers of formulas that are size-bounded by Γ with respect to canonical variable renaming. [Kin99]*

Lemma 7. *If formula ϕ is size-bounded by a set of formulas Γ, and G is a G-Rule, then $reverse\_apply\_grule(G, \phi, \Gamma)$ function will always pass a set of formulas, Φ, to the $backward\_chain()$ function, such that for each and every $\phi^* \in \Phi$, ϕ^* is also size-bounded by Γ. [Kin99]*

Lemma 8. *If all formulas in Γ are size-bounded by a finite set Γ^0, and ϕ matches no G-Rule's conclusion, then for every substitution σ such that*

$$\sigma \in backward\_chain\_one(\phi, \Gamma) \tag{3.12}$$

$\sigma\phi$ is size-bounded by Γ^0. [Kin99]

Lemma 9. *If Γ is size-bounded by a finite set Γ^0, and Φ is the set of premises of some S-Rule S, then $backward\_chain(\Phi, \Gamma)$ must halt. [Kin99]*

Lemma 10. *If all formulas in Γ are size-bounded by a finite set Γ^0, and S is an S-Rule, then the formulas returned by $apply\_srule(S, \Gamma)$ are size-bounded by Γ^0. [Kin99]*

Lemma 11. *If all formulas in $fringe$ and Γ are size-bounded by some finite set Γ^0, then $closure(fringe, \Gamma)$ must halt. [Kin99]*

Theorem 5 (*theory_gen*() Termination [Kin99])**.** *If the prerequisites in Definition 25 hold, then the theory_gen() function in Algorithm 1 always terminates.*

Theorem 6 (*derivable*() Termination)**.** *If the prerequisites in Definition 25 hold, then the derivable() function in Algorithm 7 always terminates.*

3.5 Decidability

Having presented the soundness, completeness and termination properties of theory generation, we summarize the following decidability results.

Theorem 7 (*theory_gen*() is a decision procedure). *Let $\Gamma^{\#}$ be the (R, S_Rules) representation of the theory generated from the set of grounded formulas Γ^0 using R, where,*

$$R = \{S\_Rules \cup G\_Rules \cup W\_Rules\} \tag{3.13}$$

Then theory_gen() Algorithm 1 is a decision procedure to determine if a grounded formula is a member of $\Gamma^{\#}$.

Since *theory_gen*() is sound, complete and terminates in finite time, this leads us to the following claim:

Corollary 1 (*derivable*() is a decision procedure). *Let the partial theory representation $(\emptyset, \Gamma^{\#})$ be the result generated after function closure$(\Gamma^0, \emptyset)$ Algorithm 2 terminates using a set of rules of inference R. Let ϕ be a grounded formula we wish to query for validity. $\Gamma^0 \nvdash \phi$ iff:*

$$\emptyset = backward\_chain(\phi, \Gamma^{\#}) \tag{3.14}$$

3.6 Summary

In Chapter 2, we gave a rigorous introduction to language L, the rules of inference, the *theory_gen*() and *derivable*() algorithms and their syntactical constraints. We showed the soundness, completeness and termination properties of the theory generation algorithms in this chapter. These properties allow one to prove or disprove a grounded query formula in finite time. Theory generation will be used as a subroutine in our PTGPA protocol analysis framework. Leveraging these desired properties, we show in Chapter 4 that algorithms of PTGPA are decision procedures for determining correctness of protocols under Alpha-S and Beta-S.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

The Framework of PTGPA

Contents

4.1 Introduction

This chapter introduces the Pre-conditioned Theory Generation Protocol Analyzer (PT-GPA). PTGPA is a general-purpose framework that automates analyses of protocols. It uses theory generation as a subroutine to determine the validity of security assertions (formally modeled as formulas in language L) before each message exchange and at the completion of a protocol. When a protocol and its assertions are provided, PTGPA is able to determine the correctness of the protocol without further manual intervention.

PTGPA requires a set of pre-conditions to be specified for the messages exchanged at each step in a protocol. Before each protocol step, theory generation is performed using knowledge that each individual principal has gathered up to this point. Once theory representation is generated, validity of each pre-condition can be checked using the *derivable*() function in Algorithm 7.

PTGPA allows a protocol designer to specify a set of post-conditions. Unlike the pre-conditions, validities of post-conditions are checked once when an entire protocol runs to completion. Therefore post-conditions resemble the set of goals expected at the completion of a protocol. Theory generation is used to carry out verification of post-conditions.

Given a protocol $\mathfrak{D}$, its pre-conditions Ω and post-conditions Σ, we discuss what constitutes correctness under two scenarios. The first scenario is when PTGPA is applied to a given $\mathfrak{D}$ without modification. The objective is to check the correctness of the protocol under *no* external attacks. The protocol runs to end without any messages having been altered or added. This scenario is known as the Alpha Scenario (Alpha-S). Roughly speaking, $\mathfrak{D}$ is correct under Alpha-S if it runs to completion, Ω is valid before each message exchange and Σ is valid at completion.

The second scenario, known as the Beta Scenario (Beta-S), deals with a modified protocol $\mathfrak{D}^*$ from $\mathfrak{D}$, obtained using a controlled process that adopts Dolev-Yao idealization (see Definition 34). In the modified protocol $\mathfrak{D}^*$, intruders are modeled explicitly as principals that participate in message exchanges. $\mathfrak{D}^*$ can thus be thought of as a specific attack rendered on $\mathfrak{D}$. We will define what it means for $\mathfrak{D}$ to be correct under a specific Beta-S, which is an attack scenario $\mathfrak{D}^*$ on $\mathfrak{D}$.

We start this chapter by giving relevant definitions. We next present the main PTGPA algorithm and explain how theory generation techniques from Chapter 2 are applied. Finally, the properties of PTGPA are briefly justified. Proofs for Chapter 4 are in Appendix B. The PTGPA analysis framework is an original contribution of this thesis.

4.2 Definitions

In this section, we present relevant definitions used in the PTGPA framework.

Definition 29 (Principal). *Principals are any entities that send or receive messages in a protocol.*

The messages exchanged in a protocol are modeled as transmissions. A transmission consists of a sender, a receiver and a message body.

Definition 30 (Transmission $I_A \rightarrow I_B : X$). *Let I_A and I_B be two principals. A transmission, denoted as $I_A \rightarrow I_B : X$, is a formula that is defined as I_A sending message X to I_B, where message X is a term in Definition 2 which is* grounded.

We will assume all transmissions complete instantly: as soon as a message is sent by I_A, it is received by I_B. Both I_A and I_B are specific principals, and X is always a specific message that is grounded. Therefore, a message transmission is always a grounded formula when used in a concrete protocol specification.

Definition 31 (Protocol $\mathfrak{O}$). *Let I be a set of principals. A protocol $\mathfrak{O}$ of size n, is a* path *of finite length n, formed from a finite sequence of vertices of I such that, from each of its vertices there is a transmission (Definition 30) to the next vertex:*

$$\mathfrak{O} = \{I_{start} \rightarrow I_i : m_1,\ I_i \rightarrow I_j : m_2, ..., I_k \rightarrow I_{end} : m_n\} \tag{4.1}$$

where each $\{I_{start}, I_i, I_j, ..., I_k, I_{end}\}$ is from I, and $\{m_1, ..., m_n\}$ are messages from transmission $1, ..., n$ respectively. Moreover, we can denote $\mathfrak{O}$ correspondingly as:

$$\mathfrak{O} = \{\mathfrak{O}_1, ..., \mathfrak{O}_n\} \tag{4.2}$$

Pre-conditions of a protocol consist of a sequence of sets of formulas. Each set within the sequence is a set of assertions about principals' beliefs before a transmission in a protocol. Put formally:

Definition 32 (Pre-Condition Ω). *Let $\mathfrak{O}$ be a protocol of size n. Let R be a set of rules of inference, and Γ^0 be the set of grounded formulas that are assumed to be valid before protocol $\mathfrak{O}$ starts. Pre-conditions for $\mathfrak{O}$, denoted as Ω, are a finite ordered set of size n, such that for each $\Omega_k \in \Omega$, Ω_k is a* finite *(possibly empty) set of grounded formulas, such that for any $\phi \in \Omega_k$, ϕ is expected to be valid by Definition 17, with respect to $\Gamma^0 \cup \mathfrak{O}_1 \cup ... \cup \mathfrak{O}_{k-1}$ and R.*

Pre-conditions are thus defined for each message transmission. All pre-conditions associated with a transmission must be valid before that transmission can proceed. We assume the protocol halts prematurely if there is an invalid pre-condition before a transmission k.

Next, we define post-conditions of a protocol. Post-conditions are assertions about principals' beliefs at the completion of a protocol. In our framework, we will not consider the validity of post-conditions if a protocol halts prematurely. Formally, post-conditions are defined as:

Definition 33 (Post-Condition Σ). *Let $\mathfrak{O}$ be a protocol, R be a set of rules of inference, and Γ^0 be the set of grounded formulas that are assumed to be valid before $\mathfrak{O}$ starts. Post-conditions of $\mathfrak{O}$, denoted as Σ, are a* finite *set of grounded formulas that are all expected to be valid, by Definition 17, with respect to $\Gamma^0 \cup \mathfrak{O}$ and R.*

The set of formulas in a post-condition Σ can be divided into two groups.

1. I_{end}'s assertions of the message it received from the last transmission of $\mathfrak{O}$, denoted as $\Sigma_{(+)}$. $\Sigma_{(+)}$ are all valid iff I_{end} believes it has successfully finished its role in protocol $\mathfrak{O}$ without observing any abnormalities.

2. Assertions on goals that protocol $\mathfrak{O}$ should achieve as whole when it completes, made by each and every participating honest principal who believes its role in protocol $\mathfrak{O}$ has successfully completed. We denote this set of assertions as $\Sigma_{(-)}$.

Definition 34 (Dolev-Yao Intruder I^*). *Given a protocol $\mathfrak{O}$, a Dolev-Yao Intruder, denoted as I^*, is a special principal that can read, modify, redirect and remove any* *message transmission in a protocol. A Dolev-Yao Intruder has the ordinary capabilities of a regular principal and is only limited by its knowledge of any secrets or keys and thus cryptographic operations that require those knowledge.*

Next, we define synthesization and sub-messages before introducing the attack generation procedure. In our definition of attack generation, we require any modification of an original message by an intruder to be structurally similar to the original message. This requirement is reflected in the following definition.

Definition 35 (Synthesization). *Let m be a message in any transmission in Definition 30. A synthesization of m, denoted as m', is inductively defined below:*

1. *If m is a constant, m' can be any constant.*

2. *Let f be a function. Let $m = f(t_1, ..., t_n)$, where $n > 0$, and $\{t_1, ..., t_n\}$ be terms. Then:*

$$m' = f(t'_1, ..., t'_n) \tag{4.3}$$

is a synthesization of m iff for any i, term t'_i is a synthesization of t_i.

There are restrictions on what message synthesizations an intruder principal I^* can construct given an original message m. If m is a constant, then I^* can construct its synthesization by replacing it with any other constant of I^*'s knowledge. If m is a keyed function f (i.e. a keyed encryption or hash function) and I^* does not know the key, then the only possible message synthesizations that I^* can construct are m itself or any pre-computed synthesizations of m in I^*'s memory (I^* learned these synthesizations of m from other principals or from history). If I^* knows the key, or f is not a keyed function; let $m = f(t_1, ..., t_n)$, then I^* can construct any synthesization of m, denoted as $m' = f(t'_1, ..., t'_n)$, only if for any $i \leq n$, I^* can construct any synthesization t'_i of t_i.

Definition 36 (Sub-Message). *Let m be a message in any transmission in Definition 30; m is a sub-message of itself. Moreover, if m is of the form $m = f(m_1, ..., m_n)$*

for some function symbol f with arity n where $n > 0$, then sub-messages of each of $\{m_1, ..., m_n\}$ are sub-messages of m.

Having defined intruder, synthesization and sub-messages, we are now ready to define the process of attack generation.

Definition 37 (Attack Generation $\mathbb{A}$). *Let I be the set of principals in an original protocol $\mathfrak{O}$, Ω and Σ be the pre-conditions and post-conditions of $\mathfrak{O}$. An attack on $\mathfrak{O}$ is obtained through a transformation function $\mathbb{A}$:*

$$\mathbb{A} : \{\mathfrak{O}, \Omega, \Sigma\} \to \{\mathfrak{O}^*, \Omega^*, \Sigma^*\} \tag{4.4}$$

where $\mathfrak{O}^*, \Omega^*, \Sigma^*$ are respectively, the resulting attack protocol, an instance of Definition 31; pre-conditions of $\mathfrak{O}^*$ in Definition 32; and post-conditions of $\mathfrak{O}^*$ in Definition 33.

$\mathbb{A}$ is defined as follows: Let $\{\mathfrak{O}^*, \Omega^*, \Sigma^*\}$ be initially empty. For k going from 1 to the size of $\mathfrak{O}$,

1. *Denote $\mathfrak{O}_k = I_i \to I_j : m_k$; the corresponding $\mathfrak{O}^*_k$, is one of the following 4 cases, as long as $\{\mathfrak{O}^*_1, ..., \mathfrak{O}^*_k\}$ is a valid path, and any messages exchanged are constructable by their senders:*

   (a) *<u>A path from I_i to I_j</u>: $\{I_i \to I^*_1 : m_k,\ I^*_1 \to I^*_2 : m^2_k,\ ...,\ I^*_{n-1} \to I_j : m^n_k\}$, where $\{I^*_1, ..., I^*_{n-1}\}$ are any principals including intruders and excluding I, and m^n_k is a synthesization of m_k, or*

   (b) *<u>A path from I_i to I^*_n</u>: $\{I_i \to I^*_1 : m_k,\ I^*_1 \to I^*_2 : m^2_k,\ ...,\ I^*_{n-1} \to I^*_n : m^n_k\}$, where $\{I^*_1, ..., I^*_n\}$ are any principals including intruders and excluding I, or*

   (c) *<u>A path from I^*_0 to I_j</u>: $\{I^*_0 \to I^*_1 : m^1_k,\ I^*_1 \to I^*_2 : m^2_k,\ ...,\ I^*_{n-1} \to I_j : m^n_k\}$, where $\{I^*_0, ..., I^*_{n-1}\}$ are any principals including intruders and excluding I, and m^n_k is a synthesization of m_k, or*

   (d) *<u>An empty path</u> $\emptyset$.*

The attack protocol $\mathfrak{O}^*$ is then expanded:

$$\mathfrak{O}^* = \mathfrak{O}^* \cup \mathfrak{O}^*_k \tag{4.5}$$

Additionally, in cases (1a) or (1c), for any sub-message m^{sub} of m_k that is semantically referred to in $\mathfrak{O}_{k+1}, ..., \mathfrak{O}_n$, in $\Omega_{k+1}, ..., \Omega_n$ and in Σ, m^{sub} is replaced with the corresponding sub-term in m^n_k. $\mathfrak{O}$, Ω and Σ are updated for the later iterations of $\mathbb{A}$.

2. *The pre-conditions in this attack protocol at this step k, denoted as Ω^*_k are:*

   (a) *In cases (1a) or (1b), $\Omega^*_k = \{\overbrace{\Omega_k, \emptyset, ..., \emptyset}^{n \text{ in total}}\}$,*

   (b) *In case (1c), $\Omega^*_k = \{\overbrace{\emptyset, ..., \emptyset}^{n \text{ in total}}\}$*

   (c) *In case (1d), $\Omega^*_k = \emptyset$*

   The pre-conditions Ω^* are expanded:

$$\Omega^* = \Omega^* \cup \Omega^*_k \tag{4.6}$$

After the for-loop exits, when $k > n$, the post-condition Σ^* is computed as follows [1]*: For any $\phi \in \Sigma$, only if ϕ is an assertion on any honest principal, for each and every of whose transmissions sent or received in the original protocol, there is a corresponding transmission included in $\mathfrak{O}^*$, then:*

$$\Sigma^* = \Sigma^* \cup \phi \tag{4.7}$$

Definition 38 (Intruder Assertion Υ). *Let $\mathfrak{O}^*$ be an attack of size n, R be a set of rules of inference and Γ^0 be the set of grounded formulas that are assumed to be valid before*

[1]Note that ϕ can be from either $\Sigma_{(-)}$ or $\Sigma_{(+)}$. It is then put into the corresponding category in Σ^*, either $\Sigma^*_{(-)}$ or $\Sigma^*_{(+)}$

$\mathfrak{O}^*$ starts. The intruder assertion, denoted as Υ, is a finite *set of grounded formulas, such that for each $\phi \in \Upsilon$:*

1. *ϕ is expected to be valid, in Definition 17, with respect to $\Gamma^0 \cup \mathfrak{O}_1^* \cup ... \cup \mathfrak{O}_k^*$ and R, for any k such that $0 \leq k \leq n$, and,*
2. *ϕ is an assertion of an intruder principal's belief in attack $\mathfrak{O}^*$*

Subsection 4.2.1 gives an example based on Definition 37 and Definition 38.

4.2.1 Example Transformations

In Definition 37, we introduced the procedure for characterizing an attack, given an original protocol. We defined a controlled transformation procedure from an original protocol as well as its pre- and post-conditions to that of any of its attacks. This transformation saves protocol designers the effort to come up with updated sets of pre- and post-conditions each time when analyzing new attacks. The procedure is simple yet expressive. We now give some example transformations.

Example Transformation

An example of an original protocol between Alice and Bob and its derived attacks involving intruder Eva, is presented in Figure 4-1. The original protocol $\mathfrak{O}$ includes at least two message transmissions between Alice and Bob. Attack $\mathfrak{O}^1$ models an eavesdropping attack. For each message transmission $A \to B : X$ in $\mathfrak{O}$, it is replaced with $A \to Eva \to B : X$. The same message is forwarded by Eva. Neither Alice nor Bob is aware of the presence of the intruder. Attack $\mathfrak{O}^2$ models a man-in-the-middle attack. It is similar to $\mathfrak{O}^1$ except the messages passed on from Eva are synthesizations of the corresponding message in $\mathfrak{O}$ that Eva is able to construct. In attack $\mathfrak{O}^3$, Eva tries to impersonate Bob. Eva prevents Bob from receiving any messages sent by Alice. It then replies to Alice with message synthesizations that it can construct. Notice that because m_2' is different from m_2 in $\mathfrak{O}$, Alice can reply to Eva with m_3' that is different from m_3 in $\mathfrak{O}$.

Original $\mathfrak{D}$
$Alice \rightarrow Bob : m_1$
$Bob \rightarrow Alice : m_2$
$Alice \rightarrow Bob : m_3$
$Bob \rightarrow Alice : m_4$
$\ldots$

(a) Original protocol $\mathfrak{D}$

Attack $\mathfrak{D}^1$
$Alice \rightarrow Eva : m_1$
$Eva \rightarrow Bob : m_1$
$Bob \rightarrow Eva : m_2$
$Eva \rightarrow Alice : m_2$
$\ldots$

(b) Eavesdropper attack

Attack $\mathfrak{D}^2$
$Alice \rightarrow Eva : m_1$
$Eva \rightarrow Bob : m_1'$
$Bob \rightarrow Eva : m_2$
$Eva \rightarrow Alice : m_2'$
$\ldots$

(c) Man-in-the-middle

Attack $\mathfrak{D}^3$
$Alice \rightarrow Eva : m_1$
$Eva \rightarrow Alice : m_2'$
$Alice \rightarrow Eva : m_3'$
$Eva \rightarrow Alice : m_4'$
$\ldots$

(d) Impersonation attack

Figure 4-1: An original protocol $\mathfrak{D}$ and examples of its derived attack protocols $\mathfrak{D}^1$, $\mathfrak{D}^2$ and $\mathfrak{D}^3$ using transformation function $\mathbb{A}$ in Definition 37.

Pre- and Post-Conditions

In 2a and 2b of Definition 37, we do not impose any pre-conditions for any message transmissions that are sent by an intruder or by any honest principals not involved in the original protocol and whose involvement in the attack protocol is because of the intruders. This relaxation allows us to model stronger intruders, since imposing these pre-conditions will likely to cause detections of the attack. However, we leave open the possibility of imposing a set of appropriate pre-conditions in these message transmission paths.

Moreover, we require that the transformed post-conditions Σ^* only contain assertions from principals whose action of sending or receiving a message in $\mathfrak{D}$ is included in $\mathfrak{D}^*$. This means that the set of transformed post-conditions for principals who did not participate completely in an attack protocol is dropped. Consider the following example protocol and its attack in Figure 4-2:

Suppose that one of the post-condition in $\mathfrak{D}$ is that Bob requires m_n to be fresh. Since post-conditions are meant to reflect non-intruders' assertions at the completion of

Original $\mathfrak{O}$	Attack $\mathfrak{O}^*$
$Alice \rightarrow Bob : m_1$	$Alice \rightarrow Bob : m_1$
$Bob \rightarrow Alice : m_2$	$Bob \rightarrow Eva : m_2$
$Alice \rightarrow Bob : m_3$	$Eva \rightarrow Alice : m'_2$
...	$Alice \rightarrow Eva : m'_3$
...	...
$Alice \rightarrow Bob : m_n$	$Alice \rightarrow Eva : m'_n$
(a) Original protocol $\mathfrak{O}$	(b) Attack scenario

Figure 4-2: An original protocol $\mathfrak{O}$ and its attack $\mathfrak{O}^*$ in which Bob participated first but was prevented from receiving subsequent transmissions due to intruder Eva.

a protocol and Bob did not participate fully in the attack protocol. Bob only received m_1 as expected. He did not receive m_n. The post-conditions of the attack protocol will not include any of Bob's assertions, such as the freshness requirement on m_n.

Alice participated fully in this attack protocol as expected. That is, for any transmission in $\mathfrak{O}$ in which Alice is either a sender or a receiver, there is a corresponding message transmission in $\mathfrak{O}^*$ such that Alice is the same sender or receiver. Alice is not aware of Eva's existence and expects to communicate to Bob. Alice's post-conditions need to be validated to ensure that Alice is not fooled by Eva.

4.3 Correctness: Alpha-S and Beta-S

In this section, we define what constitutes correctness given a protocol and realization(s) of its attacks through the transformation procedure in Definition 37. The correctness analysis is twofold. First, a protocol is correct under Alpha-S iff the set of pre-conditions for each transmission are valid and the post-conditions are valid at the protocol's completion. Alpha-S correctness is defined from the perspective of non-intruder principals. Formally, this is:

Definition 39 (Alpha-S Correctness). *Let $\mathfrak{O}$ be an original protocol of size n, with pre-conditions Ω and post-conditions Σ. Let Γ^0 be the set of grounded formulas that are initially valid before the protocol starts. Let R be a set of rules of inference. $\mathfrak{O}$ is* *correct under Alpha-S with respect to Γ^0 and R, iff:*

1. *For any k and ϕ^{Pre} such that $1 \leq k \leq n$ and $\phi^{Pre} \in \Omega_k$,*

$$\Gamma^0 \cup \mathfrak{O}_1 \cup ... \cup \mathfrak{O}_{k-1} \vdash \phi^{Pre} \tag{4.8}$$

   and,

2. *For any formula $\phi^{Post} \in \Sigma$:*

$$\Gamma^0 \cup \mathfrak{O} \vdash \phi^{Post} \tag{4.9}$$

Alpha-S is the primary verification scenario. It tests to see if a given candidate protocol design satisfies a collection of security goals and business requirements at each step and at completion. If a given candidate protocol is Alpha-S correct, we can move on to validate the protocol's correctness under various attack scenarios, a set of Beta-S that addresses the protocol designer's interests.

Correctness of an original protocol under Beta-S is more involved, and is defined from the perspectives of both non-intruder principals as well as intruders. First, intruders may cause certain pre-conditions Ω or some post-conditions in $\Sigma_{(+)}$ to fail. In this case, the attack is said to be *detected*, and we need to ensure all intruder assertions are valid. That is, some honest principal has recognized the abnormalities from an attack. We need to ensure that the intruder has not gained any advantages at the time of detection. For example, we may want to assert that the intruder never sees the plain-text content of an encrypted message sent by an honest principal and we will validate this assertion even the attack is detected.

Otherwise, the protocol runs to the end without non-intruder principals detecting any abnormalities. We still need to check the intruder assertions as well as all post-conditions to make sure non-intruder principals are not fooled. An example of this scenario is an eavesdropper that listens to all messages. The non-intruder principals do not detect any abnormalities. We however need to show that the eavesdropper did not obtain any information it is not supposed to obtain. Put formally, correctness under Beta-S is defined as follows:

Definition 40 (Beta-S Correctness). *Let $\mathfrak{D}^*$ of size n be an attack of protocol $\mathfrak{D}$, obtained through $\mathbb{A}$ in Definition 37. Let Σ^* be the post-conditions of $\mathfrak{D}^*$, Ω^* be the pre-conditions of $\mathfrak{D}^*$ and Υ be the set of grounded formulas of intruder assertions. Let Γ^0 be the set of grounded formulas that are initially valid before the $\mathfrak{D}^*$ starts. Let R be a set of rules of inference. $\mathfrak{D}$ is correct under a specific attack $\mathfrak{D}^*$ with respect to Γ^0 and R iff:*

1. *If for some k and ϕ^{Pre} such that $1 \leq k \leq n$ and $\phi^{Pre} \in \Omega^*_k$,*

$$\Gamma^0 \cup \mathfrak{D}^*_1 \cup ... \cup \mathfrak{D}^*_{k-1} \nvdash \phi^{Pre} \tag{4.10}$$

   Then for any formulas $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{D}^*_1 \cup ... \cup \mathfrak{D}^*_{k-1} \vdash \phi^{Intruder} \tag{4.11}$$

2. *If for some ϕ^{Last} such that $\phi^{Last} \in \Sigma^*_{(+)}$ and,*

$$\Gamma^0 \cup \mathfrak{D}^* \nvdash \phi^{Last} \tag{4.12}$$

   Then for any formulas $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{D}^* \vdash \phi^{Intruder} \tag{4.13}$$

3. *Otherwise, for any $\phi^{Post} \in \Sigma^*_{(-)}$:*

$$\Gamma^0 \cup \mathfrak{D}^* \vdash \phi^{Post} \tag{4.14}$$

and, for any $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{O}^* \vdash \phi^{Intruder} \tag{4.15}$$

4.4 PTGPA Algorithms

Having defined Alpha-S and Beta-S correctness, this section presents the corresponding PTGPA algorithms. PTGPA allows one to verify the correctness of a given protocol specification with full automation. The main PTGPA consists of PTGPA-Alpha and PTGPA-Beta algorithms, respectively targeting Definitions 39 and 40. PTGPA uses theory generation as its subroutine. Proofs and disproofs of any formulas in preconditions, post-conditions and intruder assertions are conducted using the *theory_gen*() function in Algorithm 1 and *derivable*() function in Algorithm 7.

We first define an auxiliary function *allValid*() to determine if each and every grounded formula in a given set Φ is valid with respect to a theory representation $\Gamma^{\#}$ and a set of rules of inference R. The function is defined in Figure 4-3.

```
Algorithm 8: $allValid(\Phi, \Gamma^{\#}, R)$
Data: Set of grounded formulas $\Phi$, theory representation $\Gamma^{\#}$, rules of inference
      R.
Result: Return true if every formula from a given set $\Phi$ is valid with respect to
        $\Gamma^{\#}$ and R; Otherwise return false
for each $\phi \in \Phi$ do
    if $derivable(\phi, \Gamma^{\#}) = \emptyset$ then
        return false;
    end
end
return true;
```

Figure 4-3: allValid: To determine if every formula in a given set Φ is valid with respect to a theory representation $\Gamma^{\#}$ and a set of rules of inference R.

4.4.1 PTGPA-Alpha Algorithm

Given a protocol $\mathfrak{O}$, its initial assumptions Γ^0 and its pre-conditions and post-conditions, as well as a set of rules of inference R, the PTGPA-Alpha algorithm returns a boolean to signify if $\mathfrak{O}$ is correct under Alpha-S. The algorithm is outlined in Figure 4-4.

```
Algorithm 9: $PTGPA\_Alpha(\mathfrak{O}, \Gamma^0, \Omega, \Sigma, \preceq, R)$
Data: Protocol $\mathfrak{O}$ of size n, initial assumption $\Gamma^0$, pre-conditions $\Omega$,
      post-conditions $\Sigma$, Preorder $\preceq$, rules of inference $R$ built using $\preceq$
Result: Return true if $\mathfrak{O}$ is correct under Alpha-S; false otherwise
for $k \leftarrow 1$ to $n$ do
    $\Gamma^k = theory\_gen(\Gamma^{k-1} \cup \mathfrak{O}_{k-1}, R, \preceq)$ ;
    if $allValid(\Omega_k, \Gamma^k, R) = false$ then
        return false
    end
end
$\Gamma^{\#} = theory\_gen(\Gamma^n \cup \mathfrak{O}_n, R, \preceq)$ ;
if $allValid(\Sigma, \Gamma^{\#}, R) = false$ then
    return false
end
return true;
```

Figure 4-4: PTGPA-Alpha Algorithm: To determine if a given protocol is correct under Alpha-S.

Algorithm 9 generates a theory representation Γ_k prior to each message transmission k. All pre-conditions for that transmission are validated against Γ_k. The theory representation $\Gamma^{\#}$ at protocol completion is computed and post-conditions are validated against it.

4.4.2 PTGPA-Beta Algorithm

Given an attack $\mathfrak{O}^*$ of protocol $\mathfrak{O}$, obtained through $\mathbb{A}$ in Definition 37, initial assumptions Γ^0, pre-conditions in Definition 32, post-conditions in Definition 33, intruder assertions in Definition 38, as well as a set of rules of inference R, the PTGPA-Beta algorithm returns a boolean to signify if $\mathfrak{O}$ is Beta-S correct under attack $\mathfrak{O}^*$. The algorithm is outlined in Figure 4-5.

```
Algorithm 10: PTGPA_Beta($\mathfrak{O}^*, \Gamma^0, \Omega^*, \Sigma^*, \Upsilon, \preceq, R$)
Data: $\mathfrak{O}$'s Attack $\mathfrak{O}^*$ of size n, initial assumptions $\Gamma^0$, pre-conditions $\Omega^*$,
      post-conditions $\Sigma^*$, intruder assertions $\Upsilon$, preorder $\preceq$, rules of inference $R$
Result: Return true if $\mathfrak{O}$ is correct under this specific Beta-S; false otherwise
for $k \leftarrow 1$ to $n$ do
    $\Gamma^k = theory\_gen(\Gamma^{k-1} \cup \mathfrak{O}^*_{k-1}, R, \preceq)$ ;
    if allValid($\Omega^*_k$,$\Gamma^k$,$R$)=false then
        return allValid($\Upsilon$,$\Gamma^k$,R)
    end
end
$\Gamma^\# = theory\_gen(\Gamma^n \cup \mathfrak{O}^*_n, R, \preceq)$ ;
if allValid($\Sigma^*_{(+)}$,$\Gamma^\#$,$R$)=false then
    return allValid($\Upsilon$,$\Gamma^\#$,R)
end
if allValid($\Sigma^*_{(-)}$,$\Gamma^\#$,$R$)=false then
    return false
end
if allValid($\Upsilon$,$\Gamma^\#$,$R$)=false then
    return false
end
return true
```

Figure 4-5: PTGPA-Beta Algorithm: To determine if a given protocol is correct under Beta-S.

Algorithm 10 generates a theory representation Γ_k prior to each message transmission k. All pre-conditions for that transmission are validated. If any pre-condition fails (attack detected), we return whether all intruder assertions are valid. Otherwise the protocol runs to completion. Let $\Gamma^\#$ be the theory representation computed at protocol completion. We check if all formulas in $\Sigma^*_{(+)}$ are valid. If any is invalid (attack detected), we return whether all intruder assertions are valid. Otherwise, we return if each and every formula from post-conditions in $\Sigma^*_{(+)}$ and intruder assertions Υ is valid.

4.5 Decidability of PTGPA

We now show Algorithms 9 and 10 are decision procedures for determining correctness of a candidate protocol in Alpha-S and Beta-S respectively. First we state the proof of termination. We start by claiming termination of the $allValid()$ function.

4.5.1 Termination

Lemma 12 (allValid()). *If Φ is a finite set of grounded formulas and $\Gamma^{\#}$ is a theory representation in Definition 22 with respect to a set of rules of inference R (including W, G and S rules), then the $allValid(\Phi, \Gamma^{\#}, R)$ function in Algorithm 8 terminates in finite time.*

Theorem 8 (PTGPA Termination). *PTGPA-Alpha Algorithm 9 and PTGPA-Beta Algorithm 10 terminate in finite time.*

4.5.2 Decidability

To prove the decidability of the PTGPA algorithms we present the following two lemmas.

Lemma 13 (allValid() is a decision procedure). *Given a set of grounded formulas Φ, a theory representation $\Gamma^{\#}$ generated from a set of initial assumptions Γ^0 and a set of rules of inference R, allValid(Φ,$\Gamma^{\#}$,R) returns true iff each and every formula in Φ is valid with respect to Γ^0 and R.*

Lemma 14 (Rolling-Equivalence). *Let R be a set of rules of inference, Φ^k be an ordered set of k-1 grounded formulas such that $\Phi^k = \{\phi_1, ...\phi_{k-1}\}$. Let Γ^k be the theory representation constructed from $theory\_gen(\Phi^k, R, \preceq)$; and Γ^k_* be the theory representation constructed from $theory\_gen(\Gamma^{k-1} \cup \phi_{k-1}, R, \preceq)$, then:*

$$\Gamma^k \equiv \Gamma^k_* \tag{4.16}$$

Having constructed lemmas 13 and 14, we are now ready to justify the decidability of the PTGPA Alpha-S in Algorithm 9 and PTGPA Beta-S in Algorithm 10.

Theorem 9 (Alpha-S PTGPA is a decision procedure). *Given a protocol $\mathfrak{O}$, its initial assumptions Γ^0 and its pre-conditions Ω and post-conditions Σ, as well as a set of rules* *of inference R, $\mathfrak{O}$ is Alpha-S correct in Definition 39, iff the PTGPA-Alpha algorithm in Algorithm 9 returns true.*

Theorem 10 (Beta-S PTGPA is a decision procedure). *Given an attack protocol $\mathfrak{O}^*$, its initial assumptions Γ^0, pre-conditions Ω^*, post-conditions Σ^*, intruder assertions Υ obtained partly* [2] *using $\mathbb{A}$ in Definition 37 from an original protocol $\mathfrak{O}$, its initial assumptions Γ^0, pre-conditions Ω and post-conditions Σ, $\mathfrak{O}$ is Beta-S correct under attack $\mathfrak{O}^*$ in Definition 40, iff the PTGPA-Beta algorithm in Algorithm 10 returns true.*

4.6 Java Implementation

We have implemented the theory generation framework and the two PTGPA algorithms in Java. Our implementation is compact and is a little under 1000 lines of Java. The implementation focused mainly on the functionality and was not optimized for performance. We have successfully analyzed a number of protocols and their attacks with this implementation. The details of these experiments are discussed in chapters 6, 7 and 8.

[2]The initial assumptions Γ^0 for the attack protocol is manually specified.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

The *TGPay* Deductive System

Contents

5.1 Introduction

This chapter formally introduces *TGPay*, a self-contained proof system that complies with the prerequisites of theory generation. *TGPay* is a "little logic" for payment protocols. We start with a sequence of definitions and then expand on a set of core semantics that are frequently used in secure payment protocols.

We provide a concrete definition of preorder $\preceq_{TGPay}$ that satisfies Definition 11. With this specific $\preceq_{TGPay}$ definition, we are able to partition the entire set of rules of inference from *TGPay* into G-Rules, S-Rules or W-Rules, a prerequisite mandated by theory generation in Definition 25.

For each S-Rule, we label its primary premises with α and side conditions (non-primary premises) with β. For example, let ϕ_1 be a primary premise and ϕ_2 be a side-condition and C be a conclusion. Applying this annotation, we express this S-Rule as:

$$\frac{\begin{array}{c}\alpha : \phi_1 \\ \beta : \phi_2\end{array}}{C} \tag{5.1}$$

Since a G-Rule does not have any primary premises, all premises of a G-Rule will be annotated using β. So a typical G-Rule is annotated as:

$$\frac{\begin{array}{c}\beta 1 : \phi_1 \\ \beta 2 : \phi_2\end{array}}{C} \tag{5.2}$$

We will not annotate any W-Rules since they are size-preserving by definition.

The remainder of this chapter is organized as follows: We first describe each predicate and function used in the $TGPay$ deductive system. Where possible, we adopt notations that are consistent with those established in literature. We then present a concrete definition of preorder $\preceq_{TGPay}$ and we use it to define the set of rules in $TGPay$. We provide a summary at the end of this chapter to briefly justify that $TGPay$ meets the prerequisites of theory generation in Definition 11.

5.2 Functions and Predicates

5.2.1 General Semantics

Definition 41 (Belief: $P \models X$). *Let P be a principal and X be a variable. Define predicate $P \models X$ to denote P's current belief over the truth of X.*

The predicate $\models$ is central to the proof system. Notice that if $P \models X$, then it is not necessary that X is thought valid by everyone. X is merely principal P's individual belief. One cannot assume that $(P \models X) \models (Q \models X)$, since P and Q will be exposed to different sets of knowledge.

Additionally, our notion of beliefs is stable. That is, once $P \models X$ is derived, it holds for the rest of the protocol analysis. The protocols we analyze are short-lived, with a typical lifetime measured in seconds.

Finally, we assume that all principals are capable of reasoning about their beliefs. This also applies to intruders.

Definition 42 (Trustworthy: $P \mapsto \star$). *Let P be a principal. Define predicate $P \mapsto \star$ to denote that P is trustworthy. Notice that $\star$ is just a decoration not a variable.*

For example, suppose $\aleph$ is a certificate authority and is considered trustworthy. Let P be a principal. P's belief of $\aleph$ being trustworthy can be formalized in *TGPay* as $P \models \aleph \mapsto \star$.

Definition 43 (Message Concatenation: $X.Y$). *Let variables X and Y be messages. Let X.Y denote the message concatenation function that appends Y at the rear of X. This function has left-to-right associativity.*

Definition 44 (Message Inclusion: $\varpi(X,Y)$). *Let variables X and Y be messages. Define predicate $\varpi(X,Y)$ and let it return true iff message Y is part of a larger message X.*

5.2.2 Time and Freshness

Definition 45 (Freshness: $\sharp(X)$). *Let variable X be a message. Define predicate $\sharp(X)$ to denote that message X is recently created.*

A message is considered fresh iff it was generated recently so that it is sufficient to rule out any replay attack.

Definition 46 (Time Interval: $\Theta(T_a, T_b)$). *Let T_a, T_b be two times such that $T_a < T_b$. Define $\Theta(T_a, T_b)$ to be a time interval $[T_a, T_b]$. Notice that Θ is not a function but a decoration.*

Definition 47 (Current Time Interval: $\triangle(\Theta(T_a, T_b))$). *Define predicate $\triangle(\Theta(T_a, T_b))$ to denote that time interval $[T_a, T_b]$ brackets the current time.*

An example of the application of $\triangle(T_a, T_b)$ is in a digital certificate, where it is used to denote the life span of an issued certificate.

Definition 48 (Time-stamp of Construction: $\Theta(T)$). *Define $\Theta(T)$ to denote that message T is a time stamp of message construction. Θ is not a function but a decoration.*

Definition 49 (Time-stamp of Expiration: $\Theta^*(T)$). *Define $\Theta^*(T)$ to denote that message T is a time stamp of message expiration. Θ^* is not a function but a decoration.*

Definition 50 (Recent Time-stamp: $\triangle(\Theta(T))$). *Define predicate $\triangle(\Theta(T))$ to denote that T represents a recent time stamp.*

Definition 51 (Non-Expired Time-stamp: $\triangle(\Theta^*(T))$). *This is predicate overloading. Define predicate $\triangle(\Theta^*(T))$ to denote that T represents a time stamp of an expiration that has not been reached.*

Definition 52 (Nonce: $\varkappa(N)$). *Let N be a string. Define $\varkappa(N)$ to mean that N is intended to be used as a nonce. Notice that $\varkappa()$ is not a function but a decoration.*

5.2.3 Public Key Infrastructure (PKI)

In this subsection we define predicates and symbols for PKI. Some of our notations for PKI are consistent with [SY08].

Definition 53 (Identity: $\delta(X)$). *Let X be a variable. Define $\delta(X)$ to mean X represents an identity of a principal. $\delta()$ is not a predicate but merely a decoration.*

Definition 54 (Public Key: $\wp(P, K)$). *Let variable P be a principal and K be a public key of an RSA cryptography scheme [RSA78]. $\wp(P, K)$ denotes that principal P is associated with a public key K.*

Definition 55 (Private Key: $\psi(P, K)$). *Let variable P be a principal and K be a private key of an RSA cryptography scheme. $\psi(P, K)$ denotes that principal P is associated with a private key K.*

Definition 56 (Public Key Encryption: $\pi_P(X, Q)$). *Let variable X be a message and Q be a principal. Define function $\pi_P(X, Q)$ to denote the cipher-text corresponding to X encrypted (exponentiation performed) using the* public key *of principal Q.*

Definition 57 (Private Key Encryption: $\pi_V(X, Q)$). *Let variable X be a message and Q be a principal. Define function $\pi_V(X, Q)$ to denote the cipher-text corresponding to X encrypted (exponentiation performed) using the* private key *of principal Q.*

A certificate, denoted as Ψ, consists of a plaintext part Ψ^T and a cryptographic signature part Ψ^*.

Definition 58 (X.509 Certificate: $\Psi(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph)$). *An X.509 certificate [AF99] $\Psi(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph)$ of principal P issued by authority $\aleph$, is defined as:*

$$\Psi^T(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph).\Psi^*(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph) \tag{5.3}$$

Definition 59 (X.509 Plain-Text Portion: $\Psi^T(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph)$).

$$\Psi^T(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph) = \delta(ID).\Theta(T_1^P, T_2^P).\wp(P, K_P).\aleph \tag{5.4}$$

$\Psi^T(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph)$ is the plain-text portion of the certificate Ψ. It contains the issuer $\aleph$, issuee P, time stamps with issuing and expiring time T_1^P and T_2^P, issuee's public key K_P as well as certificate identity, ID.

Definition 60 (X.509 Signature Portion: $\Psi^*(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph)$). *This is the cryptographic signature of the hash (Definition 67) of all the information in the X.509 plain-text portion in Definition 59:*

$$\Psi^*(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph) = \pi_V(\theta(\Psi^T(P, \delta(ID), \Theta(T_1^P, T_2^P), K_P, \aleph)), \aleph) \tag{5.5}$$

where $\theta()$ is a hash function in Definition 67.

A certificate revocation list (CRL) is a list of certificates (or more specifically, a list of serial numbers for certificates) that have been revoked, and therefore should not be relied upon [IETF12b]. A certificate is said to be valid iff it is not on a CRL and is defined below:

Definition 61 (Certificate Validity: $\chi(\delta(ID))$). *Let variable ID be a certificate identity and define predicate $\chi(\delta(ID))$ to denote that the certificate associated with this ID has* not *been revoked.*

5.2.4 Authentication

Definition 62 (Recipient: $\mathfrak{R}(X, P)$). *Let variable X be a message and P be a principal. Predicate $\mathfrak{R}(X, P)$ returns true iff message X has intended recipient P.*

Definition 63 (Originator: $\mathfrak{S}(X, P)$). *Let variable X be a message and P be a principal. Predicate $\mathfrak{S}(X, P)$ returns true iff message X originated from P, or loosely speaking, P once uttered X.*

$\mathfrak{S}(X, P)$ merely states that P said X, or bound its identity to X. It is possible that many principals bound their identities to the same message X.

Definition 64 (Not Originator $\mathfrak{S}^{-1}(X, P)$). *Let variable X be a message and P be a principal. Predicate $\mathfrak{S}^{-1}(X, P)$ returns true iff P has never uttered X in any instance of a protocol.*

Definition 65 (Symmetric Key: $\gamma_K(K, P, Q)$). *Let K be a symmetric key and P and Q be two principals. Define predicate $\gamma_K(K, P, Q)$ and let it return true iff principals P and Q currently share an established symmetric key K. Moreover, K is shared only between P and Q.*

Definition 66 (Symmetric Encryption: $\pi_S(X, P, Q)$). *Let X be a message and P and Q be two principals. Define function symbol $\pi_S(X, P, Q)$ to denote the ciphertext corresponding to encrypting plaintext X with the symmetric key shared between principals P and Q.*

5.2.5 Hash and Message Authentication Code

Definition 67 (Message Digest: $\theta(X)$). *Let variable X represent a message. Define function $\theta(X)$ to be the digest of message X computed using a cryptographic hash function [IETF12c] [IETF12d]. We assume perfect hashing. That is,*

1. $(X_1 \neq X_2) \models (\theta(X_1) \neq \theta(X_2))$,
2. *Given $\theta(X)$, it is impossible to reconstruct X,*
3. *Let f be any function, then $(X \neq f(X)) \models (\theta(X) \neq \theta(f(X)))$*

Definition 68 (Hash-based Message Authentication Code (HMAC): $\omega(X, K)$). *Let X be a message and K be a key. Define $\omega(X, K)$ to be the HMAC of message X when key K is used [IETF12a]:*

$$\omega(X, K) = \theta(K.\theta(K.X)) \tag{5.6}$$

Definition 69 (Integrity: $\eta(X, P)$). *Let variable X be a message and P be a principal. Define predicate $\eta(X, P)$ and let it return true iff X has not been altered since its original construction (announcement) by principal P. Said differently, it was exactly X that P once uttered. If P has never uttered X, the predicate will always return false.*

5.2.6 Non-Repudiation

We define a generic provability operator $\sqsupset()$ as follows:

Definition 70 (Provability: $\sqsupset(P, X)$). *Let P be a principal and X be a message. Define predicate $\sqsupset(P, X)$ and let it return true iff P can prove the validity of X to any principal other than P itself.*

5.2.7 Message Reconstruction

Definition 71 (Message Sent: $P \trianglerighteq X$). *Let P be a principal and X be a message. Define predicate $P \trianglerighteq X$ and let it return true iff P once sent message X to some other* *principal. That is, X was revealed by P and hence some other principals potentially received X.*

Definition 72 (Message Received: $P \trianglelefteq X$). *Let P be a principal and X be a message. Define predicate $P \trianglelefteq X$ and let it return true iff P once received message X. It is not necessarily known who the message constructor or sender was.*

Definition 73 (Potential Reconstruction: $\sqcap(P, X)$). *Let P be a principal and variable X be a message. Define predicate symbol $\sqcap(P, X)$ to indicate that principal P is able to reconstruct X.*

Notice predicate $\sqcap$ denotes a principal's potential or feasibility to reconstruct a message from information it generated or received. The principal may or may not have actually constructed the message at time of the predicate evaluation. For example, a principal has received message X, and therefore it is able to reconstruct $X.\theta(X)$, although it does not have to explicitly compute and store $X.\theta(X)$.

5.3 Preorder $\preceq_{TGPay}$

5.3.1 Definitions

This section defines the preorder $\preceq_{TGPay}$ used in our proof system $TGPay$. The definition of $\preceq_{TGPay}$ is motivated by [Kin99]. We have extended it to include additional forms of atomic variables. In this section, we first introduce the relevant definitions. The claims of the properties of the preorder will be presented next. Proofs to all lemmas and theorems are presented in Appendix C.

Definition 74 (Atomic Variable [Kin99]). *The atomic variables of functions and predicates are those arguments (variables) that are constants (0-ary functions) under* any *substitutions.*

For example, consider the predicate $\sharp(X)$, in which X is a non-atomic variable. One can replace X with functions of any arity such as *hello.world*, $\theta(hello.world)$, etc. Now consider a different predicate $\wp(P, K)$. Its substitutions (instantiations) are not allowed to produce $\wp(Alice.Bob, \theta(key))$ in which P is replaced with a function concatenation of two constants and K is replaced with a hash function. It must be instantiated as $\wp(Alice, key_x)$ for some specific principal name "Alice" (a single constant) and an instance of a key key_x (a single constant). In *TGPay*, any variables that appear in functions and predicates that describe a principal name or key are atomic variables. Atomic variables play important roles in the preorder specification. In TGPay, the atomic variables are:

Definition 75 (Atomic Variable in $\preceq_{TGPay}$). *Any principal names are atomic variables. Any cryptographic keys, including symmetric keys and public/private keys are atomic variables. Additionally, any variable used within a decoration symbol is atomic. These include:*

1. *Time-stamp T for construction: $\Theta(T)$*

2. *Time-stamp T for expiration: $\Theta^*(T)$*

3. *Time envelope $[T_a, T_b]$: $\Theta(T_a, T_b)$*

4. *Principal ID: $\delta(ID)$*

5. *Nonce N: $\varkappa(N)$*

We are now ready to describe $\preceq_{TGPay}$. For convenience, we first define two auxiliary functions $\Pi()$ and $\Lambda()$.

Definition 76 ($\Pi(\phi)$). *Let ϕ be a formula. Define function $\Pi(\phi)$ to be the total number of predicates, functions and non-atomic variables appearing in ϕ, excluding any atomic variables.*

Definition 77 ($\Lambda(\phi, X)$). *Let ϕ be a formula and X be a variable name. Define function $\Lambda(\phi, X)$ to be the number of occurrences of X in ϕ.*

Definition 78 ($\preceq_{TGPay}$ [Kin99]). *Let ϕ_1 and ϕ_2 be two formulas in $TGPay$.*

$$\phi_1 \preceq_{TGPay} \phi_2 \tag{5.7}$$

is well defined with respect to finite sets of predicate symbols (each with fixed and finite arity), function symbols (each with fixed and finite arity) and constants, iff:

1. *Alphabetic Count:* $\Pi(\phi_1) \leq \Pi(\phi_2)$, *and*

2. *Variable Frequency:* $\Lambda(\phi_1, X) \leq \Lambda(\phi_2, X)$, *for any non-atomic variable X appearing in ϕ_1*

One observation obtained directly from Definition 74 is that substitutions performed over atomic variables do not change the *size* of the formula that the atomic variables reside in. This can be seen from the lemma below.

Lemma 15. *Let ϕ be a formula and let σ be any substitution over atomic variables from ϕ. Then,*

$$\Pi(\phi) = \Pi(\sigma\phi) \tag{5.8}$$

and

$$\Lambda(\phi, X) = \Lambda(\sigma\phi, X) \tag{5.9}$$

for any non-atomic variables X appearing in ϕ.

5.3.2 Properties of $\preceq_{TGPay}$

Next, properties of $\preceq_{TGPay}$ are discussed. We first show that the $\preceq_{TGPay}$ is a preorder operator.

Lemma 16. *$\preceq_{TGPay}$ is a preorder.*

We now show that operator $\preceq_{TGPay}$ from Definition 78 strictly satisfies monotonicity, substitution preservation and finiteness properties in Definition 11.

Monotonicity

Lemma 17 (Monotonicity [Kin99]). *Let ϕ be a formula, T_1, T_2 be terms and X be a variable from ϕ. Let substitutions $\sigma_1 = \{X \rightarrow T_1\}$ and $\sigma_2 = \{X \rightarrow T_2\}$ Then,*

$$(T_1 \preceq_{TGPay} T_2) \models (\sigma_1\phi \preceq_{TGPay} \sigma_2\phi) \tag{5.10}$$

Substitution Preservation

Lemma 18 (Substitution Preservation [Kin99]). *Let ϕ_1, ϕ_2 be two formulas, T be a term and let X be a variable from ϕ_1 or ϕ_2 or both. Let substitution $\sigma = \{X \rightarrow T\}$. Then,*

$$(\phi_1 \preceq_{TGPay} \phi_2) \models (\sigma\phi_1 \preceq_{TGPay} \sigma\phi_2) \tag{5.11}$$

Finiteness

Lemma 19 (Finiteness [Kin99]). *Let ϕ^* be any formula composed from a finite set of predicate symbols (each with finite arity), a finite set of function symbols (each with finite arity) and a finite set of constants and non-atomic variables, then the set $\{\phi | \phi \preceq_{TGPay} \phi^*\}$ must be finite modulo variable renaming.*

Theorem 11. *$\preceq_{TGPay}$ is an instance of the preorder $\preceq$ in Definition 11.*

5.4 Rules of Inference

In this section, we formally introduce the set of rules of inference in our proof system. All rules are defined with respect to the preorder in Definition 78. We start with the set of general rules and move on to rules that target specific requirements of payment protocols.

5.4.1 General Rules

The first rule is a rewrite rule on message concatenation of two sub-messages. Let X and Y be two terms.

W-Rule 1 (Rewrite - 2 Term Composition).

$$\frac{X.Y}{Y.X} \tag{5.12}$$

We also have rewrite rules for symmetric key encryption:

W-Rule 2 (Rewrite - Symmetric Key).

$$\frac{\gamma_K(K, P, Q)}{\gamma_K(K, Q, P)} \tag{5.13}$$

W-Rule 3 (Rewrite - Symmetric Encryption).

$$\frac{\pi_S(X, P, Q)}{\pi_S(X, Q, P)} \tag{5.14}$$

We have two rules about multi-part messages:

S-Rule 1 (Sub-Message Belief). *If principal P believes a composed message then it believes each of the message's sub-components.*

$$\frac{\alpha : P \models (X.Y)}{P \models X} \tag{5.15}$$

G-Rule 1 (Component Rule). *The G-Rule requires no premises and only has a conclusion. It states that message X is part of a larger message of the form X concatenated* *with Y.*

$$\frac{\beta : .}{P \models \varpi(X.Y, X)} \tag{5.16}$$

5.4.2 Time and Freshness

G-Rule 2 (Freshness - Composition). *This rule states that if P believes that message X is fresh, then P believes that a composed message X.Y is also fresh.*

$$\frac{\beta : P \models \sharp(X)}{P \models \sharp(X.Y)} \tag{5.17}$$

Note that although the composed message X.Y must be recently generated, message Y may or may not be fresh.

S-Rule 2 (Jurisdiction with Time-stamp). *This S-Rule states that if P believes principal Q is trustworthy ($\beta 2$), and if principal P believes that Q has sent a message (α) that has a fresh time-stamp ($\beta 1$ e.g. $\triangle(\Theta(T))$) with message X, then P believes X.*

$$\frac{\begin{array}{c}\alpha : P \models \mathfrak{S}(\Theta(T).X, Q) \\ \beta 1 : P \models \triangle(\Theta(T)) \\ \beta 2 : P \models Q \mapsto \star\end{array}}{P \models X} \tag{5.18}$$

S-Rule 3 (Jurisdiction with Expiration). *This S-Rule states that if P believes principal Q is trustworthy ($\beta 2$), and if principal P believes that Q has sent a message (α) with an expiration time-stamp $\Theta^*(T)$ with message X and the expiration time-stamp $\Theta^*(T)$ has not been reached ($\beta 1$), then P believes X.*

$$\frac{\begin{array}{c}\alpha : P \mid\equiv \mathfrak{S}(\Theta^*(T).X, Q) \\ \beta 1 : P \mid\equiv \Delta(\Theta^*(T)) \\ \beta 2 : P \mid\equiv Q \mapsto \star\end{array}}{P \mid\equiv X} \tag{5.19}$$

G-Rule 3 (Freshness - Public Key Encryption). *This G-Rule states that freshness of X is maintained under public key encryption. If P believes X is fresh ($\beta 1$) and P knows Q's private key ($\beta 2$), then P believes the encrypted X using Q's public key is also fresh.*

$$\frac{\begin{array}{c}\beta 1 : P \mid\equiv \sharp(X) \\ \beta 2 : P \mid\equiv \psi(Q, K_Q^{-1})\end{array}}{P \mid\equiv \sharp(\pi_P(X, Q))} \tag{5.20}$$

Note that P does not necessarily believe that X came from Q. Moreover, P and Q are likely to be the same principal.

G-Rule 4 (Freshness - Private Key Encryption). *This G-Rule states that freshness of X is maintained under private key encryption. If P believes X is fresh ($\beta 1$) and P knows Q's public key ($\beta 2$), then P believes the encrypted X using Q's private key is also fresh.*

$$\frac{\begin{array}{c}\beta 1 : P \mid\equiv \sharp(X) \\ \beta 2 : P \mid\equiv \wp(Q, K_Q)\end{array}}{P \mid\equiv \sharp(\pi_V(X, Q))} \tag{5.21}$$

G-Rule 5 (Freshness - Symmetric Key Encryption). *This G-Rule states that freshness of X is maintained under symmetric key encryption.*

$$\frac{\begin{array}{c}\beta 1 : P \mid\equiv \sharp(X) \\ \beta 2 : P \mid\equiv \gamma_K(K, P, Q)\end{array}}{P \mid\equiv \sharp(\pi_S(X, P, Q))} \tag{5.22}$$

5.4.3 PKI

S-Rule 4 (Certificate Validation Rule). *This is an S-Rule. It allows principal P to construct its belief in Q's public key from the content of Q's certificate C (α). Instrumental in the process is the validation of C's expiration ($\beta 1$) and revocation ($\beta 4$). Additionally, P must check the authenticity of C using C's issuer's public key $K_{\aleph}$ ($\beta 2$, $\beta 3$).*

$$\frac{\begin{array}{c} \alpha : P \models P \trianglelefteq \Psi(Q, \delta(ID), \Theta(T_1^Q, T_2^Q), K_Q, \aleph) \\ \beta 1 : P \models \triangle(\Theta(T_1^Q, T_2^Q)) \\ \beta 2 : P \models \wp(\aleph, K_{\aleph}) \\ \beta 3 : P \models \aleph \mapsto \star \\ \beta 4 : P \models \chi(\delta(ID)) \end{array}}{P \models \wp(Q, K_Q)} \tag{5.23}$$

S-Rule 5 (Certificate ID Extraction). *With the same set of premises, P can reconstruct the ID of Q from Q's certificate.*

$$\frac{\begin{array}{c} \alpha : P \models P \trianglelefteq \Psi(Q, \delta(ID), \Theta(T_1^Q, T_2^Q), K_Q, \aleph) \\ \beta 1 : P \models \triangle(\Theta(T_1^Q, T_2^Q)) \\ \beta 2 : P \models \wp(\aleph, K_{\aleph}) \\ \beta 3 : P \models \aleph \mapsto \star \\ \beta 4 : P \models \chi(\delta(ID)) \end{array}}{P \models \sqcap (P, \delta(ID))} \tag{5.24}$$

S-Rule 6 (Part of Certificate). *With the same set of premises, P believes this specific*

ID is part of Q's certificate.

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \Psi(Q, \delta(ID), \Theta(T_1^Q, T_2^Q), K_Q, \aleph) \\ \beta 1 : P \models \triangle(\Theta(T_1^Q, T_2^Q)) \\ \beta 2 : P \models \wp(\aleph, K_\aleph) \\ \beta 3 : P \models \aleph \mapsto \star \\ \beta 4 : P \models \chi(\delta(ID))\end{array}}{P \models \varpi(\Psi(Q, \delta(ID), \Theta(T_1^Q, T_2^Q), K_Q, \aleph), \delta(ID))} \tag{5.25}$$

5.4.4 Authentication

Authentication is the identification of the sources and destinations of messages: which principal said X and which principal is supposed to receive X. In general, authentication of a message X's announcer P (P once said X) does not imply that P currently believes X. Moreover, over an insecure channel, an established belief that P said X from a specific message transmission does not entail that subsequent messages received are also from P. Hence authentication must be carried out on a per-message basis in protocol analyses.

S-Rule 7 (Origin: Private Key Encryption). *This S-Rule states that if principal P believes that it received a message X encrypted with Q's private key (α), and that P believes that public key K_P is associated with principal Q (β), then P believes that it was principal Q who once uttered X.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_V(X, Q) \\ \beta : P \models \wp(Q, K_P)\end{array}}{P \models \mathfrak{S}(X, Q)} \tag{5.26}$$

S-Rule 8 (Destination: Public Key Encryption). *This S-Rule states that if principal P believes that it received a message that was encrypted using P's public key (α), and P has a corresponding private key K_P^{-1} (β), then P believes that message X was intended* *for P to read.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_P(X, P) \\ \beta : P \models \psi(P, K_P^{-1})\end{array}}{P \models \mathfrak{R}(X, P)} \tag{5.27}$$

S-Rule 9 (Origin: Symmetric Key). *This S-Rule states that if principal P believes that it received a message that can be successfully decrypted into X by using a symmetric key K (α1), and that P shares K with another principal Q (β); and additionally P believes that it has never uttered the cipher-text (so that P is not receiving back the same message itself sent before) at the time of evaluation (α2), then P believes that it must be principal Q who once uttered X.*

$$\frac{\begin{array}{c}\alpha 1 : P \models P \trianglelefteq \pi_S(X, P, Q) \\ \alpha 2 : P \models \mathfrak{S}^{-1}(\pi_S(X, P, Q), P) \\ \beta : P \models \gamma_K(K, P, Q)\end{array}}{P \models \mathfrak{S}(X, Q)} \tag{5.28}$$

Notice that $\mathfrak{S}(X, Q)$ should be interpreted as "it was Q who constructed or uttered X". The ciphertext $\pi_S(X, P, Q)$ may be received and relayed by principals other than Q along the way to P.

S-Rule 10 (Destination: Symmetric Key). *This S-Rule states that if principal P believes that it received a message that it can be successfully decrypted into X by using a symmetric key K (α1), and that P shares K with another principal Q (β), and additionally P believes that it has never uttered the cipher-text at the time of evaluation* *(α2), then P believes that message X was intended for P.*

$$\frac{\begin{array}{c}\alpha 1 : P \models P \trianglelefteq \pi_S(X, P, Q) \\ \alpha 2 : P \models \mathfrak{S}^{-1}(\pi_S(X, P, Q), P) \\ \beta : P \models \gamma_K(K, P, Q)\end{array}}{P \models \mathfrak{R}(X, P)} \tag{5.29}$$

S-Rule 11 (Partial Source Authentication). *This S-Rule states that if P believes that Q has once uttered a composed message X.Y then P also believes that Q once uttered a partial message X.*

$$\frac{\alpha : P \models \mathfrak{S}(X.Y, Q)}{P \models \mathfrak{S}(X, Q)} \tag{5.30}$$

S-Rule 12 (Partial Destination Authentication). *This S-Rule states that if P believes that it is the intended recipient of a composed message X.Y, then P also believes that it is the intended recipient of a partial message X.*

$$\frac{\alpha : P \models \mathfrak{R}(X.Y, Q)}{P \models \mathfrak{R}(X, Q)} \tag{5.31}$$

5.4.5 Hashing, MAC and Integrity

A message is integral if it has not been altered or modified since its announcement. In practice, message integrity is often achieved by accompanying message X with its digest or HMAC. We define relevant rules for integrity in this subsection.

S-Rule 13 (Integrity: Component). *This S-Rule states that if message X.Y has not being tampered with since its announcement, then P believes that its sub-message X is also intact (See Definition 69).*

$$\frac{\alpha : P \models \eta(X.Y, Q)}{P \models \eta(X, Q)} \tag{5.32}$$

S-Rule 14 (Integrity: Generic Hash). *This S-Rule states that if principal P believes that Q once uttered message X.Y and Y is the correct message digest of X, then P believes that principal Q once uttered X as is.*

$$\frac{\alpha : P \models \mathfrak{S}(X.\theta(X), Q)}{P \models \eta(X, Q)} \tag{5.33}$$

S-Rule 15 (Authentication: Hashed Signature, Public Key Encryption). *This S-Rule models classical digital signatures. If P believes that it can reconstruct X ($\beta 1$), it has received a message of the form $\pi_V(\theta(X), Q)$ (α), and that K_Q is some principal Q's public key ($\beta 2$), then P concludes that principal Q once uttered X.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_V(\theta(X), Q) \\ \beta 1 : P \models \sqcap (P, X) \\ \beta 2 : P \models \wp(Q, K_Q)\end{array}}{P \models \mathfrak{S}(X, Q)} \tag{5.34}$$

S-Rule 16 (Integrity: Hashed Signature, Public Key Encryption). *This rule is similar to the one above, but with a different conclusion. If P believes that it has received a message of the form $\pi_V(\theta(X), Q)$ (α) and it can reconstruct X ($\beta 1$), and that K_Q is some principal Q's public key ($\beta 2$), then P concludes message X was not modified and was the original X when it was uttered by Q.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_V(\theta(X), Q) \\ \beta 1 : P \models \sqcap (P, X) \\ \beta 2 : P \models \wp(Q, K_Q)\end{array}}{P \models \eta(X, Q)} \tag{5.35}$$

Rules 15 and 16 allow one to derive beliefs about the originator of a message and its integrity. Principals often sign the hash of a message X rather than the message itself. This signature together with the corresponding X in plain-text allows the receiver to determine authenticity. The following two S-Rules 17 and 18 are similar to rules 15 and 16. They derive the same set of conclusions provided that the hash of message X is first authenticated.

S-Rule 17 (Authentication: Signed Hashed Signature). *If P believes Q once uttered a hash of a message X (α) and P believes P can reconstruct X (β), then P concludes that Q once uttered X.*

$$\frac{\begin{array}{c}\alpha : P \models \mathfrak{S}(\theta(X), Q) \\ \beta : P \models \sqcap (P, X)\end{array}}{P \models \mathfrak{S}(X, Q)} \tag{5.36}$$

S-Rule 18 (Integrity: Signed Hashed Signature). *If P believes Q once uttered a hash of a message X (α) and P believes P can reconstruct X (β), then P concludes it was exactly X that Q once uttered. X is intact.*

$$\frac{\begin{array}{c}\alpha : P \models \mathfrak{S}(\theta(X), Q) \\ \beta : P \models \sqcap (P, X)\end{array}}{P \models \eta(X, Q)} \tag{5.37}$$

S-Rule 19 (Authentication: HMAC). *This S-Rule models authentication with an HMAC. If P believes that it has received an HMAC $\omega(X, K)$ (α1), and it can reconstruct X (β1), and that K is a key shared only between P and some principal Q (β2), and that P knows that it did not reveal $\omega(X, K)$ (α2), then P concludes that it was principal Q who once* *uttered X.*

$$\frac{\begin{array}{c}\alpha 1 : P \models P \trianglelefteq \omega(X, K) \\ \alpha 2 : P \models \mathfrak{S}^{-1}(\omega(X, K), P) \\ \beta 1 : P \models \sqcap (P, X) \\ \beta 2 : P \models \gamma_K(K, P, Q)\end{array}}{P \models \mathfrak{S}(X, Q)} \tag{5.38}$$

S-Rule 20 (Integrity: HMAC). *This S-Rule models integrity with an HMAC and has the same set of premises as Rule 19. If P believes that it has received X's HMAC ($\alpha 1$) and it can reconstruct X ($\beta 1$), and that K is a key only shared between P and some principal Q ($\beta 2$), and that P knows that itself did not utter X ($\alpha 2$), then P concludes that it was precisely this X that Q once uttered. X is intact.*

$$\frac{\begin{array}{c}\alpha 1 : P \models P \trianglelefteq \omega(X, K) \\ \alpha 2 : P \models \mathfrak{S}^{-1}(\omega(X, K), P) \\ \beta 1 : P \models \sqcap (P, X) \\ \beta 2 : P \models \gamma_K(K, P, Q)\end{array}}{P \models \eta(X, Q)} \tag{5.39}$$

5.4.6 Non-Repudiation Extension

Sometimes it is insufficient for a principal itself to form certain beliefs. For payment protocols, the need for accountability must also be addressed. Accountability reflects a principal's ability to prove certain claims to an arbitrator R (a neutral and honest principal) without assuming any knowledge of R other than its ability to reason logically. When P presents a proof to R, the proof can be transferable or non-transferable.

A transferable proof is a proof of a certain claim X, that after the claim is presented to R, R is able to use it to prove X to another honest principal R' where $R \neq R'$.

Non-transferable proofs are those proofs P constructs to convince Q about a certain claim without revealing anything other than the veracity of the statement. Consequently Q is unable to prove it to another principal. An example of a non-transferable proof is the possession of one's private key. Given a random challenge nonce generated by Q, P can return the nonce signed by its private key. Q subsequently checks the returned message with P's public key to verify. During the proof procedure P does not reveal its private key yet it is able to convince Q about P's possession of P's private key. Q cannot convince arbitrator R that P has the corresponding private key, partly because Q cannot prove to R that the challenge nonce used was fresh.

Next we focus on a small subset of accountability inference rules - accountability of messages that a principal once uttered, using the framework of *transferable* proofs.

S-Rule 21 (Non-Repudiation: Proof of Submessage). *This S-Rule states that if P believes that Q can prove a message composition X.Y then Q can prove a sub-message X.*

$$\frac{\alpha : P \models \sqsupset(Q, X.Y)}{P \models \sqsupset(Q, X)} \tag{5.40}$$

S-Rule 22 (Non-Repudiation: Private Key Encryption). *This S-Rule states that if principal P believes that it received a message that corresponds to Q encrypting X using its private key (α), and additionally P can prove that K is the public key of Q (β), then P believes that it can prove that it was Q who once uttered X.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_V(X, Q) \\ \beta : P \models \sqsupset(P, \wp(Q, K_Q))\end{array}}{P \models \sqsupset(P, \mathfrak{S}(X, Q))} \tag{5.41}$$

S-Rule 23 (Non-Repudiation: Signature Origin). *This S-Rule states that if principal P believes that it received a message that is Q's signature over the hash of message X (α), and P believes that it can reconstruct message X ($\beta 1$), and P believes that it can prove that K_Q is the public key of Q ($\beta 2$), then P believes that it can prove that it was*

Q who once uttered X.

$$\frac{\begin{array}{c}\alpha : P \mid\equiv P \trianglelefteq \pi_V(\theta(X), Q)\\ \beta 1 : P \mid\equiv \sqcap(P, X)\\ \beta 2 : P \mid\equiv \beth(P, \wp(Q, K_Q))\end{array}}{P \mid\equiv \beth(P, \mathfrak{S}(X, Q))} \tag{5.42}$$

S-Rule 24 (Non-Repudiation: Signature Integrity). *This S-Rule has the same set of premises of Rule 23; it states that if principal P believes it received a message that is Q's signature over the hash of message X (α), and P believes that it can reconstruct message X ($\beta 1$), and P believes it can prove that K_Q is the public key of Q ($\beta 2$), then P believes that it can prove that X was not modified since X was uttered by Q.*

$$\frac{\begin{array}{c}\alpha : P \mid\equiv P \trianglelefteq \pi_V(\theta(X), Q)\\ \beta 1 : P \mid\equiv \sqcap(P, X)\\ \beta 2 : P \mid\equiv \beth(P, \wp(Q, K_Q))\end{array}}{P \mid\equiv \beth(P, \eta(X, Q))} \tag{5.43}$$

S-Rule 25 (Non-Repudiation: Public-Key from Certificate). *This S-Rule allows one to be able to prove another principal's public key from its certificate. The set of premises is the same to that of S-Rule 4. If Q's certificate is valid, then P not only recognizes Q's public-key but also is able to use the certificate as a transferable proof to prove Q's public-key at the time of evaluation.*

$$\frac{\begin{array}{c}\alpha : P \mid\equiv P \trianglelefteq (\Psi(Q, \delta(ID), \Theta(T_1^Q, T_2^Q), K_Q, \aleph))\\ \beta 1 : P \mid\equiv \triangle(\Theta(T_1^Q, T_2^Q))\\ \beta 2 : P \mid\equiv \wp(\aleph, K_\aleph)\\ \beta 3 : P \mid\equiv \aleph \mapsto \star\\ \beta 4 : P \mid\equiv \chi(\delta(ID))\end{array}}{P \mid\equiv \beth(P, \wp(Q, K_Q))} \tag{5.44}$$

This S-Rule models a digital certificate as a transferable proof of one's public-key. One can similarly construct S-Rules for other identities in a certificate such as expiration date, public key and issuer.

5.4.7 Jurisdiction

S-Rule 26 (Jurisdiction Rule). *This S-Rule appeared in the original BAN papers [BAN89] and [BAN90]. It states that if P believes that Q is trustworthy ($\beta 1$) then P trusts what Q currently believes (α, $\beta 2$). This rule reduces the other principal's belief to one's own belief.*

$$\begin{array}{c} \alpha : P \models \mathfrak{S}(X, Q) \\ \beta 1 : P \models Q \mapsto \star \\ \beta 2 : P \models \sharp(X) \\ \hline P \models X \end{array} \tag{5.45}$$

5.4.8 Confidentiality Extension

S-Rule 27 (Message Extraction Rule (Public Key Encryption)). *This S-Rule states that if P believes that it received a ciphertext using principal P's public key over plaintext X (α), and P has the corresponding private key (β), then P believes it received X.*

$$\begin{array}{c} \alpha : P \models P \trianglelefteq \pi_P(X, P) \\ \beta : P \models \psi(P, K_P^{-1}) \\ \hline P \models P \trianglelefteq X \end{array} \tag{5.46}$$

This rule is different from Rule 8. Rule 8 allows one to derive the principal that message X was intended for. This rule emphasizes that P has received X and therefore can "see" the content of X.

S-Rule 28 (Message Extraction Rule (Private key encryption)). *This S-Rule states that if P believes that it received a ciphertext created using principal Q's private key* *over plaintext X (α), and P has access to Q's public key (β), then P believes it received X.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_V(X, Q) \\ \beta : P \models \wp(Q, K_Q)\end{array}}{P \models P \trianglelefteq X} \tag{5.47}$$

This rule is different from Rule 7. Rule 7 allows one to derive the principal who once said X. This rule emphasizes that P has received X and therefore can "see" the content of X.

S-Rule 29 (Message Extraction Rule (Symmetric key encryption)). *This S-Rule states that if P believes that it received a ciphertext encrypted using a symmetric key between P and Q over plaintext X (α), and additionally P has access to that key (β), then P believes that it received X.*

$$\frac{\begin{array}{c}\alpha : P \models P \trianglelefteq \pi_S(X, P, Q) \\ \beta : P \models \gamma(K, P, Q)\end{array}}{P \models P \trianglelefteq X} \tag{5.48}$$

5.4.9 Message Reconstruction

Message reconstruction allows the derivation of any message combinations a principal has potential to construct. The messages may arrive from different message transmissions, encrypted or in plain-text. The principal may not have the specific message combination actually computed and stored. Message reconstruction rules are used extensively in the instantiations of non-primary premises in the S-Rules for authentication.

S-Rule 30 (Received Sub-message). *This S-Rule states that if P believes that it received a composed message of form X.Y, then it also received the submessage X.*

$$\frac{\alpha : P \models Q \trianglelefteq X.Y}{P \models Q \trianglelefteq X} \tag{5.49}$$

S-Rule 31 (Sent Sub-message). *This S-Rule states that if P believes it sent a composed message of form X.Y, then it also sent the submessage X.*

$$\frac{\alpha : P \models Q \trianglerighteq X.Y}{P \models Q \trianglerighteq X} \tag{5.50}$$

S-Rule 32 (Message Reconstruction: Sent Single Message). *This S-Rule states that if P believes it sent a message X, then P can reconstruct X.*

$$\frac{\alpha : P \models P \trianglerighteq X}{P \models \sqcap (P, X)} \tag{5.51}$$

S-Rule 33 (Message Reconstruction: Received Single Message). *This S-Rule states that if P believes it received a message X, then P can reconstruct X.*

$$\frac{\alpha : P \models P \trianglelefteq X}{P \models \sqcap (P, X)} \tag{5.52}$$

G-Rule 6 (Message Reconstruction: Composition). *This G-Rule states that if principal P believes that Q can construct X (β1) and construct Y (β2), then P believes that Q can construct X composed with Y.*

$$\frac{\begin{array}{c}\beta 1 : P \models \sqcap (Q, X) \\ \beta 2 : P \models \sqcap (Q, Y)\end{array}}{P \models \sqcap (Q, X.Y)} \tag{5.53}$$

G-Rule 7 (Message Reconstruction: Received Messages). *This G-Rule states that if principal P believes that Q can construct X (β1) and Q has received Y (β2), then P* *believes that Q can construct X composed with Y.*

$$\frac{\beta 1 : P \models \sqcap (Q, X) \quad \beta 2 : P \models Q \trianglelefteq Y}{P \models \sqcap (Q, X.Y)} \tag{5.54}$$

G-Rule 8 (Message Reconstruction: Sent Messages). *This G-Rule states that if principal P believes that Q can construct X (β1) and Q sent Y (β2), then P believes that Q can construct X composed with Y.*

$$\frac{\beta 1 : P \models \sqcap (Q, X) \quad \beta 2 : P \models Q \trianglerighteq Y}{P \models \sqcap (Q, X.Y)} \tag{5.55}$$

G-Rule 9 (Message Reconstruction: Two Received Messages). *This G-Rule states that if principal P believes that Q received X (β1) and Y (β2), then P believes that Q can construct X composed with Y.*

$$\frac{\beta 1 : P \models Q \trianglelefteq X \quad \beta 2 : P \models Q \trianglelefteq Y}{P \models \sqcap (Q, X.Y)} \tag{5.56}$$

G-Rule 10 (Message Reconstruction: Two Sent Messages). *This G-Rule states that if principal P believes that Q sent X (β1) and Y (β2), then P believes that Q can construct X composed with Y.*

$$\frac{\beta 1 : P \models Q \trianglerighteq X \quad \beta 2 : P \models Q \trianglerighteq Y}{P \models \sqcap (Q, X.Y)} \tag{5.57}$$

G-Rule 11 (Message Reconstruction: Sent and Received Messages). *This G-Rule states that if principal P believes that Q received X (β1) and once sent Y (β2), then P* *believes that Q can construct X composed with Y.*

$$\frac{\begin{array}{c}\beta 1 : P \models Q \trianglelefteq X \\ \beta 2 : P \models Q \trianglerighteq Y\end{array}}{P \models \sqcap(Q, X.Y)} \tag{5.58}$$

5.4.10 Message Transmission

S-Rule 34 (Message Send). *This S-Rule models message sending and generates P's belief about sending a message. If message X was sent from P to Q, then P believes that it has sent out X.*

$$\frac{\alpha : P \rightarrow Q : X}{P \models P \trianglerighteq X} \tag{5.59}$$

S-Rule 35 (Message Receipt). *This S-Rule models message receipt and generates Q's belief about receiving a message. If message X was sent from P to Q, then Q believes that it has received message X.*

$$\frac{\alpha : P \rightarrow Q : X}{Q \models Q \trianglelefteq X} \tag{5.60}$$

5.5 Summary

This chapter described the *TGPay* proof system in detail. We first defined a set of functions and predicates used in payment protocols. We then provided a concrete definition of preorder $\preceq_{TGPay}$ that satisfies Definition 11. With this specific $\preceq_{TGPay}$ definition, we were able to partition the entire set of rules of inference from *TGPay* into G-Rules, S-Rules or W-Rules.

The proposed *TGPay* proof system fully satisfies the set of prerequisites mandated by theory generation in Definition 25 and hence is amenable to PTGPA algorithms. In particular, we claim:

1. Operator $\preceq_{TGPay}$ satisfies Definition 11,

2. Every rule in $TGPay$ is either a G-Rule or an S-Rule or a W-Rule with respect to $\preceq_{TGPay}$,

3. Every W-Rule in $TGPay$ is size-preserving with respect to $\preceq_{TGPay}$,

4. S/G restriction holds for all S-Rules with respect to $\preceq_{TGPay}$. For each and every S-Rule S in $TGPay$, if we let P_i be a primary premise of S and S_j be a non-primary premise of S:

   (a) P_i does not unify with the conclusions of any G-Rules in $TGPay$, and,

   (b) For all possible pairs of $\{S_j, P_i\}$ of S, $S_j \preceq_{TGPay} P_i$

With these prerequisites met, we are now ready to use $TGPay$ as a concrete proof system for the automated PTGPA analysis framework described in Chapter 4.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Contactless Bankcard Payment Protocols

Contents

6.1 Introduction

Over the past years, contactless bank cards have been widely adopted in the US. [All09] estimated that approximately 90 million contactless credit cards and 130,000 contactless readers were deployed by the end of 2009. However, the payment protocols that these bankcards adopt are not all secure. In an independent study, [HBBFJ07] examined a set of 20 contactless bank cards from multiple payment organizations and major issuing banks in the US. The study found security vulnerabilities of various degrees in all 20 cards.

Heydt-Benjamin et al.'s experiment was carried out with readers from two independent manufacturers. The contactless cards communicate with the readers over the ISO14443-B transport layer [ISO08a]. Card taps were captured and messages transmitted between the card and reader were obtained from the serial port of the readers. The outputs were then rendered in the ISO7813 [ISO08b] mag-stripe format, and the payment protocols used by the bank cards were reverse engineered.

Based on the findings, the protocols used by the 20 bank cards were placed into four categories. In this chapter, we will formally model two of them in *TGPay* and show how PTGPA analyses can be carried out on the two protocols and their attacks.

After that, we present *TGPay* formalizations of two authentication protocols used in the Europay-MasterCard-Visa (EMV) protocols [EMV11a] [EMV11b]. We provide detailed analyses using the PTGPA framework.

6.2 Bank Card Protocol Overview

The bank card payment protocol starts when a contactless bank card taps on a reader. The reader forwards the card information and a transaction description string M to a back-end server, which then checks for the consistency of the information and, if satisfied, reports to a billing gateway. Since the reader and the back-end server often share a secure communication channel such as TLS [IR08a], and the reader merely forwards information it receives, we will omit modeling the reader explicitly. Instead, we assume the contactless bankcard communicates directly with the payment server, with M as part of the message transmission. Adopting this simplification, we now provide high-level descriptions of two contactless payment protocols, Type A and Type B, from [HBBFJ07].

1. Card Protocol A: This card type always sends the reader a set of static information that contains its Primary Account Number (PAN), card holder name and

expiration date. A sample serial output from a commercial reader with this type of card is presented in Figure 6-1. The first line is the message contained in the primary track of a mag-stripe card; the second line contains almost the same information, and represents the second track of a mag-stripe card.

```
Bxxxxxx6531xxxxxx^DOE/JANE^0906101000000000000000000000000858000000
                    xxxxxx6531xxxxxx=09061010000085800000
```

Figure 6-1: Message transmitted when card type A taps on a contactless reader. The top line represents the primary track and the second line represents the secondary track in the mag-stripe [HBBFJ07].

In Figure 6-1, "xxxxxx6531xxxxxx" is the PAN, "DOE/JANE" is the card holder name, and "0906" is the expiration date. Let CHD be a term that represents the concatenation of PAN, card holder name and the expiration date. We assume that the remaining string is a signature from the issuing bank over the entire card holder data (CHD). In our model, we make the simplifying assumption that the payment server Se is responsible for issuing bank cards. Se has a public-private key pair $\{k_{Se}, k_{Se}^{-1}\}$. We can formalize the message in Figure 6-1 in *TGPay* as:

$$M.CHD.\pi_V(\theta(CHD), Se) \tag{6.1}$$

which is the concatenation of plain text CHD and signature of its hash. Since the CHD is transmitted in clear, this protocol is clearly susceptible to an eavesdropping attack. Moreover, there is no freshness component sent by this type of cards - it always sends the same string when communicating with a reader. Therefore it is also susceptible to a replay attack. Last, M is not signed, so the protocol is vulnerable to cut and paste attacks.

2. Card Protocol B:

   This card type still sends the card holder data CHD in the clear. However, it has significantly enhanced its security by storing a counter value that increases monotonically for each tap. An example serial output for this type of card when communicating with a reader is presented in Figure 6-2.

Bxxxxxx1079xxxxxxˆDOE/JANEˆ09011011000000000001000000000000
xxxxxx1079xxxxxx=090110110000**016***0022*1

Figure 6-2: Message transmitted when card type B taps on a contactless reader. The top line represents the primary track and the second line represents the secondary track in the mag-stripe card format [HBBFJ07].

In this sample output, "0022" in italics is a counter value that increases at every tap. This provides freshness for every transaction message. The three preceding digits ("016" in bold) change at every tap, and they are assumed to be a signature over the CHD and the counter value. In this case, we assume the payment server keeps a mapping of each CHD to its latest counter value the server has seen. Moreover, the payment server shares a unique symmetric key k with each card that is identified by the CHD. We can formalize the message in Figure 6-2 in $TGPay$ as:

$$M.CHD.C.\omega(CHD.C, k) \tag{6.2}$$

where C is a specific counter value and $\omega(CHD.C, k)$ is an HMAC over $CHD.C$ using key k.

6.3 Card A Alpha-S Analysis

Since the card A protocol always sends the same static message, it is vulnerable to replay attacks. One of the steps the payment server takes, upon receiving the message from the contactless card and the transaction specific payload M, is to check this information for freshness and consistency. There are three principals in this protocol: the bank card (Bc), the payment server (Se) and a billing gateway (Bi) that processes transactions for billing. We assume the channel between Se and Bi is secure and we omit modeling the contactless reader since it only appends the transaction description information M and sends everything to the payment server. The UML sequence diagram of the card type A protocol is presented in Figure 6-3.

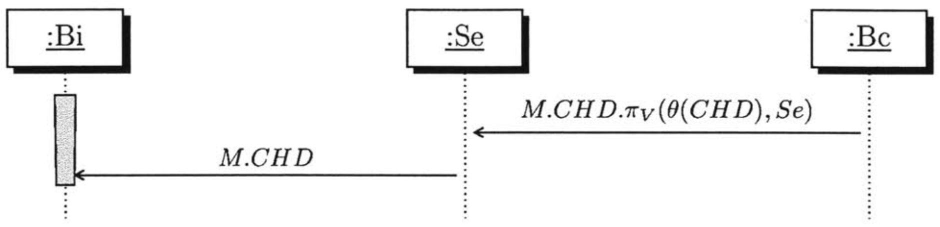


Figure 6-3: UML sequence diagram for contactless payment protocol type A.

Next we show how the protocol, as well as its initial assumptions, and its pre- and post-conditions are formalized in *TGPay*. Last, we show how the PTGPA analysis is carried out to verify the protocol's correctness.

6.3.1 Initial Assumption

The initial assumptions are Se's recognition of its own public and private key. In *TGPay*, they are modeled as two grounded formulas:

$$Se \models \wp(Se, K_{Se}) \tag{6.3}$$

$$Se \models \psi(Se, K_{Sh}^{-1}) \tag{6.4}$$

6.3.2 Pre- and Post-conditions

The set of pre-conditions is the collection of security goals that are expected to be achieved before each message exchange step. We provide a possible list of goals and briefly discuss how they can be formalized in *TGPay*[1]. We focus on the set of pre-conditions that Se imposes before communicating with the billing gateway Bi. Suppose

[1]There are many other significant goals that we do not address in this example.

Se has received $M.CHD.\pi_V(\theta(CHD), Se)$ as in Figure 6-3. A number of security requirements must be met before Se sends a message to the billing server.

1. Se needs to believe that CHD represents information of a card that was issued by Se. We model this in *TGPay* by stating Se once uttered CHD:

$$Se \models \mathfrak{S}(CHD, Se) \tag{6.5}$$

2. Se needs to ensure that CHD is intact since it was uttered (manufactured) by Se. It was exactly this CHD that Se uttered:

$$Se \models \eta(CHD, Se) \tag{6.6}$$

3. Se needs to ensure that the transaction $M.CHD$ was intended to be made by Bc. The transaction, represented by the goods purchased M and the financial instrument used to pay CHD, was initiated by BC and not by some other principal. We model this in *TGPay* as:

$$Se \models \mathfrak{S}(M.CHD, Bc) \tag{6.7}$$

   That is, $M.CHD$ was uttered by Bc.

4. It was exactly this transaction $M.CHD$ that Bc requested, and not some other $M.CHD'$ for some CHD':

$$Se \models \eta(M.CHD, Bc) \tag{6.8}$$

5. Se needs to ensure that the transaction request is recent. Se must be able to rule out any replay attack in which an old transaction request was recorded in an eavesdropping attack and re-rendered here. We formalize this requirement as the

following Se belief:

$$Se \mid\equiv \sharp(M.CHD) \tag{6.9}$$

In this example, for simplicity, we omit the post-conditions, which usually include payment requirements such as non-repudiation.

6.3.3 Analysis

We programmed this protocol specification and verified it using our PTGPA Java implementation. The protocol halted prematurely at the beginning of message two in Figure 6-3 due to violation of pre-conditions.

Two pre-conditions 6.5 and 6.6 were proved valid. They are $Se \mid\equiv \mathfrak{S}(CHD, Se)$ and $Se \mid\equiv \eta(CHD, Se)$. However, the protocol failed to justify pre-conditions 6.7, 6.8 and 6.9: $Se \mid\equiv \mathfrak{S}(M.CHD, Bc)$, $Se \mid\equiv \eta(M.CHD, Bc)$ and $Se \mid\equiv \sharp(M.CHD, Bc)$. That is, Se was not convinced that the transaction was recently initiated by BC and that the content of the transaction was intact. Specifically, 6.7 fails because message M.CHD was not signed by Bc; 6.8 fails because of the same reason; and 6.9 fails because there is no freshness component in M.CHD.

By Definition 39, this protocol is not Alpha-S correct with respect to the specified initial assumptions, pre- and post-conditions. That is, this protocol design fails to address some of the essential security requirements as reflected in 6.7, 6.8 or 6.9.

6.4 Card B Alpha-S Analysis

The protocol for card type B uses counter C and symmetric encryption as well as HMAC. We assume that Se knows the latest counter value associated with a given bank card. A UML sequence diagram of this protocol is presented at Figure 6-4:

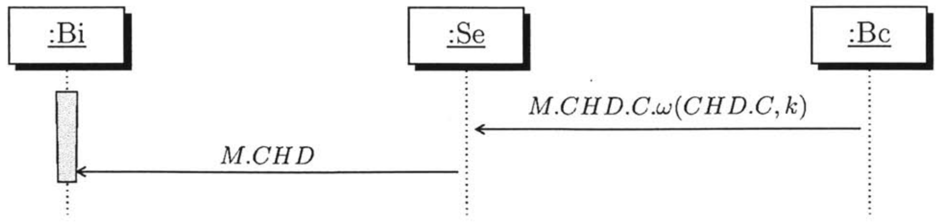


Figure 6-4: UML sequence diagram for contactless payment protocol type B.

6.4.1 Initial Assumptions

We will assume Bc shares a symmetric key k with Se. Key k is unique to Bc and Se and neither Bc nor Se uses the same k for other applications. Moreover, Se knows if the counter value C sent by Bc is fresh, since Se keeps track of the latest counter value of BC. Finally, Se believes it never uttered HMAC $\omega(CHD.C, k)$, so that Bc cannot replay it. Only Se and Bc, who are the only principals that possess k, can construct this HMAC. Se believes that keys are stored and retrieved securely on all cards. Only the protocol running on Bc can have access to k by the design of hardware security. Therefore, if Se believes it has never revealed $\omega(CHD.C, k)$, Se knows the HMAC is a non-transferable proof of Bc's authentication (It must be generated by Bc). Thus, we require the following set of initial assumptions:

$$Bc \models \gamma_K(k, Bc, Se) \tag{6.10}$$

$$Se \models \gamma_K(k, Bc, Se) \tag{6.11}$$

$$Se \models \sharp(C) \tag{6.12}$$

$$Se \models \mathfrak{S}^{-1}(\omega(CHD.C, k), Se) \tag{6.13}$$

6.10 states that Bc knows the symmetric key; 6.11 states that Se knows the symmetric key. 6.12 states that Se knows that C is fresh; and 6.13 states that Se never uttered the HMAC.

6.4.2 Pre- and Post-conditions

We now list a possible set of pre- and post-conditions. We require no pre-conditions for message one from Bc to Se in Figure 6-4. A set of pre-conditions is specified for message two, before Se reports to Bi.

1. Se needs to derive the belief that the bank card information $CHD.C$ is sent by Bc, not any other bank card Bc'. We model this requirement in $TGPay$ as:

$$Se \models \mathfrak{S}(CHD.C, Bc) \tag{6.14}$$

2. Se needs to derive the belief that $CHD.C$ is intact since Bc uttered it:

$$Se \models \eta(CHD.C, Bc) \tag{6.15}$$

3. Se needs to ensure that $M.CHD.C$ is fresh:

$$Se \models \sharp(M.CHD.C) \tag{6.16}$$

4. Se needs to ensure that it was Bc who requested transaction $M.CHD.C$:

$$Se \models \mathfrak{S}(M.CHD.C, Bc) \tag{6.17}$$

5. Se needs to ensure that transaction request $M.CHD.C$ is intact since it was uttered by Bc:

$$Se \models \eta(M.CHD.C, Bc) \tag{6.18}$$

We will specify one $\Sigma_{(-)}$ post-condition on non-repudiation. That is, Se believes that it can prove that the transaction (including the description of the goods, time-date, and the payment instrument) was requested by Bc. We can model this assertion as the following grounded formula in $TGPay$:

$$Se \models \sqsupset(Se, \mathfrak{S}(M.CHD.C, Bc)) \qquad (6.19)$$

6.4.3 Analysis

We programmed this protocol specification and verified it using our PTGPA Java implementation. The protocol halted prematurely while validating pre-conditions for step two. Three pre-conditions, 6.14, 6.15 and 6.16 were proved valid. They are, Se believes that Bc said CHD.C; Se believes the integrity of CHD.C; and Se believes that CHD.C is fresh. However, the protocol failed to justify 6.17 and 6.18. This implies that Bc's intention to send M cannot be proved. M is not signed by Bc. By Definition 39, we conclude that this protocol is not Alpha-S correct with respect to the specified initial assumptions, pre- and post-conditions. Notice that we did not validate the post-condition. This is because failures of any pre-conditions are sufficient to invalidate the candidate protocol in Alpha-S.

6.5 Card B Beta-S Analysis

In this section, we show how an attack scenario can be obtained by applying the transformation procedure in Definition 37 to the original protocol specification in Figure 6-4. We then show how Beta-S analysis can be carried out by specifying intruder assertions.

In general, we will not carry out any Beta-S analyses for any candidate protocols that are not Alpha-S correct, since the original protocols are in themselves already flawed. However, to illustrate of Beta-S, we will drop the two pre-conditions 6.17 and 6.18, as well as post-condition 6.19. We run the original protocol specification against this set of relaxed pre- and post-conditions and we find it is now Alpha-S correct. We now proceed with this set of relaxed pre- and post-conditions.

The original protocol for card type B consists of two message transmissions:

$$Bc \rightarrow Se : M.CHD.C.\omega(CHD.C, k) \tag{6.20}$$

$$Se \rightarrow Bi : M.CHD \tag{6.21}$$

Since the second transmission is assumed to be carried over a secure channel, we will focus on an attack that alters the first transmission. Using the modification strategy in Definition 37(1a), we can replace $M.CHD.C.\omega(CHD.C, k)$ with a path that starts from Bc and ends at Se, via an intruder principal I^*. Concretely, if we let I^* be a Dolev-Yao intruder, transmission 6.20 can be replaced with the following path:

$$Bc \rightarrow I^* : M.CHD.C.\omega(CHD.C, k) \tag{6.22}$$

$$I^* \rightarrow Se : M.CHD.C.\omega(CHD.C, k) \tag{6.23}$$

This path can be thought of as an eavesdropping attack that silently listens to the communication channel between Bc and Se. Since there are no message content modifications, by Definition 37, no existing pre-conditions or post-conditions are modified. However, an empty set of pre-conditions is added for message transmission 6.23. The set of pre-conditions for message transmission 6.22 remains an empty set, and the pre-conditions for 6.21 consist of: 6.14, 6.15 and 6.16.

The intruder assertion we impose is on the secrecy of the bank card information CHD. Since transmissions between contactless bank cards and readers are broadcasted, they can be read by anyone that eavesdrops. We require that eavesdroppers I^* cannot obtain the plain-text of the financial information CHD. Formally, our intruder assertion is:

$$\neg I^* \models \sqcap (I^*, CHD) \tag{6.24}$$

That is, the intruder I^* does not believe it can reconstruct CHD.

6.5.1 Analysis

We programmed this protocol specification and verified it using our PTGPA Java implementation. The protocol ran successfully to the end. There was no post-condition in this attack protocol to check. However, we found the intruder assertion is invalid. Our Java implementation correctly computed $I^* \models \sqcap (I^*, CHD)$ as a valid formula. That is, intruder I^* can in fact reconstruct CHD. This is because CHD was sent in the clear from Bc to Se. By Definition 21, intruder assertion 6.24 is invalid. By Definition 40, Beta-S correctness, we conclude that the original protocol is not correct when confronted with this specific secrecy attack.

6.5.2 Summary

In the previous sections, we used some real-world bank card payment protocols as examples for our PTGPA analyses. We showed how security requirements were formalized in *TGPay* and, through PTGPA, we identified a number of flaws in these protocols. Specifically, the Type A protocol failed to convince Se that a transaction was recently initiated by Bc and the content of the transaction was intact. The Type B protocol failed to address Bc's intention for a transaction for a specific M. The protocol is vulnerable to a cut-and-paste attack on M and violates the secrecy requirement on the card holder data CHD.

6.6 EMV Authentication Protocols

EMV [EMV11a] [EMV11b] is a leading payment standard administrated by EMVCo (http://www.emvco.com). EMV is not a specification for a single protocol. It consists of multiple highly configurable modules that target different components of payment protocols. In this section, we formalize the authentication part of the EMV protocol in *TGPay* and carry out its Alpha-S PTGPA analyses.

The authentication protocol used in EMV can be modeled using two principals, the reader Re and the bank card Bc. We assume all bank cards are issued by a trusted third principal $\aleph$. Principal $\aleph$ has a public-private key pair $\{k_\aleph, k_\aleph^{-1}\}$. The private key of $\aleph$ $k_\aleph^{-1}$ is secretly stored and the public key $k_\aleph$ is known to all readers. We assume each bank card securely stores its private k_{Bc}^{-1}.

At the initialization step, Bc and Re exchange a sequence of handshake messages to set up the protocol. (We do not model these steps) At the end of the initialization, Bc sends its CHD, including expiration date, PAN and some auxiliary information such as the Card Risk Management Data Object List (CDOL) to Re. The EMV protocol is now ready to perform card authentication.

The purpose of authentication is to verify the integrity of CHD and to check if it actually comes from the claimed Bc. The EMV specification provides three types of authentication methods: Static Data Authentication (SDA), Dynamic Data Authentication (DDA) and Combined Data Authentication (CDA). CDA is like DDA but with additional transactional information included in the authentication messages. In the remainder of this chapter, we will formalize the SDA and DDA authentication protocols in *TGPay* and then perform their PTGPA analyses.

6.6.1 Static Data Authentication

In SDA, the card returns a signature of the hash of the CHD, signed by card issuer $\aleph$. This signature was pre-computed at the time of manufacture and is stored on the card. The SDA authentication protocol can be formalized in *TGPay* as the following message transmission:

$$Bc \rightarrow Re : CHD.\pi_V(\theta(CHD), \aleph) \tag{6.25}$$

Initial Assumptions

The initial assumptions are the establishment of keys prior to the authentication. Formally, we have:

1. $\aleph$ believes its public key $K_{\aleph}$:

$$\aleph \models \wp(\aleph, K_{\aleph}) \tag{6.26}$$

2. $\aleph$ believes its private key $K_{\aleph}^{-1}$:

$$\aleph \models \psi(\aleph, K_{\aleph}^{-1}) \tag{6.27}$$

3. Re believes $\aleph$'s public key $K_{\aleph}$:

$$Re \models \wp(\aleph, K_{\aleph}) \tag{6.28}$$

4. Re believes that $\aleph$ is trustworthy:

$$Re \models \aleph \mapsto \star \tag{6.29}$$

Pre-conditions

Bc does not impose any pre-conditions before sending this message. So the pre-condition for SDA is the empty set.

Post-conditions

The set of $\Sigma_{(+)}$ post-conditions that Re requires are formalized as follows:

1. Re believes that CHD was issued by $\aleph$:

$$Re \models \mathfrak{S}(CHD, \aleph) \tag{6.30}$$

2. Re believes the integrity of CHD from $\aleph$. It was precisely CHD that $\aleph$ issued:

$$Re \models \eta(CHD, \aleph) \tag{6.31}$$

3. Re believes that it was Bc who sent this CHD:

$$Re \models \mathfrak{S}(CHD, Bc) \tag{6.32}$$

4. Re believes that CHD was recently sent by Bc and is not a replay. We model this requirement by asserting:

$$Re \models \sharp(CHD) \tag{6.33}$$

5. Re believes the integrity of CHD from Bc:

$$Re \models \eta(CHD, Bc) \tag{6.34}$$

Analysis

We programmed this protocol specification and verified it using our PTGPA Java implementation. We found that the last three post-conditions 6.32, 6.33 and 6.34 are invalid. Using SDA leaves attackers the possibility of replaying some $\pi_V(\theta(CHD), \aleph)$ to Re. Re can verify that CHD was genuinely issued by $\aleph$. However, Re cannot prove that it was a specific Bc who recently sent CHD or that CHD is intact since it was sent by that Bc.

By Definition 39, we conclude that the SDA authentication is not Alpha-S correct with respect to the specified set of initial assumptions, pre- and post-conditions. Note that EMV understands these restrictions and recommends using SDA only in limited, low risk circumstances.

6.6.2 Dynamic Data Authentication

Some of the limitations of SDA are remedied by DDA. In DDA, authentication of the card is carried out with a random challenge and application of a digital signature. (The card can also authenticate the reader; we will not model this in this example.) Bc first sends CHD and its certificate to Re. The certificate is not a standard X.509 certificate [GS91]; it is abbreviated. In *TGPay*, the certificate can be formalized as a signature over an expiration time-stamp, Bc's public key and the hash of CHD, signed by the private key of $\aleph$, $k_{\aleph}^{-1}$.

$$Bc \to Re : CHD.\pi_V(\Theta^*(T^{exp}).\wp(Bc, k_{Bc}).\theta(CHD), \aleph) \tag{6.35}$$

Re next sends a Dynamic Data Authentication Data Object (DDOL) to Bc. DDOL contains a reader-generated nonce N_{Re} that is known by Re to be fresh, plus some additional information that we will not model.

$$Re \to Bc : \varkappa(N_{Re}) \tag{6.36}$$

Upon receiving N_{Re}, Bc constructs an ICC Dynamic Data Object (ICCDDO) that consists of a freshly generated nonce N_{Bc}. Bc then returns its signature over the ICCDDO and the hash of the ICCDDO and DDOL.

$$Bc \to Re : \pi_V(\varkappa(N_{Bc}).\theta(\varkappa(N_{Bc}).\varkappa(N_{Re})), Bc) \tag{6.37}$$

The certificate in 6.35 is signed by the card issuer $\aleph$. It recognizes that the public key k_{Bc} is associated with the CHD. Therefore, the CHD is associated with anyone who can demonstrate its possession of the corresponding private key. In 6.36, a fresh challenge is sent to Bc. In 6.37, Bc provides a signature over this nonce and demonstrate that it has the matching private key, and therefore, authenticates itself to Re. We now describe formalizations of the set of initial assumptions.

Initial Assumptions

1. Bank card issuer ℵ knows its public and private keys:

$$\aleph \models \wp(\aleph, K_{\aleph}) \tag{6.38}$$

$$\aleph \models \psi(\aleph, K_{\aleph}^{-1}) \tag{6.39}$$

2. Re knows ℵ's public key:

$$Re \models \wp(\aleph, K_{\aleph}) \tag{6.40}$$

3. Bc knows N_{Bc} is fresh and Re knows N_{Re} is fresh:

$$Bc \models \sharp(\varkappa(N_{Bc})) \tag{6.41}$$

$$Re \models \sharp(\varkappa(N_{Re})) \tag{6.42}$$

4. Re knows time-stamp $\Theta^*(T^{exp})$ has not expired:

$$Re \models \triangle(\Theta^*(T^{exp})) \tag{6.43}$$

5. Re considers bank card issuer ℵ trustworthy:

$$Re \models \aleph \mapsto \star \tag{6.44}$$

Pre-Conditions

The three sets of pre-conditions for message one, two and three are respectively:

1. $\emptyset$ (empty set): Bc always starts the authentication part of the DDA protocol.

2. After receiving the first message and prior to message two, Re needs to believe that CHD was announced by ℵ and is intact and still valid. Re further needs to

believe that Bc's public key is k_{Bc}. Formally:

$$Re \models \mathfrak{S}(CHD, \aleph) \tag{6.45}$$

$$Re \models \eta(CHD, \aleph) \tag{6.46}$$

$$Re \models \wp(Bc, k_{Bc}) \tag{6.47}$$

3. $\emptyset$ (empty set): Bc always responds to DDOL.

Post-Conditions

At the end of the protocol, Re needs to believe that the response from Bc was generated recently using the private key that corresponds to the public key k_{Bc}. If this belief can be validated, then Re believes that Bc is in possession of the private key that corresponds to k_{Bc}. Hence Bc is authenticated. Formally, the $\Sigma_{(+)}$ post-condition requests that the following assertion can be derived:

$$Re \models \mathfrak{S}(\varkappa(N_{Re}), Bc) \tag{6.48}$$

That is, Bc authenticates itself by acknowledging the freshly generated nonce N_{Re}.

Analysis

We programmed this protocol specification and verified it using our PTGPA Java implementation. The protocol successfully ran to completion. All post-conditions are valid. By Definition 39, we conclude that the protocol is Alpha-S correct. We did not perform any analysis with respect to Beta-S.

6.7 Summary

In this chapter, we specified two real-world bank-card payment protocols and the EMV SDA and DDA card authentication protocols in *TGPay*. We showed how security goals were formalized and used to carry out PTGPA analyses to uncover security flaws in full automation.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Mobile-Reader Payment Protocols

Contents

7.1 Introduction

In the following two chapters, we propose two mobile contactless payment protocols that provide high security as well as additional features such as non-repudiation. We discuss their formalizations in the *TGPay* language framework and we focus on demonstrating their analyses using PTGPA.

The first mobile payment protocol allows an NFC-enabled mobile phone to pay at an NFC reader that connects securely to a payment processing gateway. The second type of mobile payment protocol, discussed in the next chapter, allows an NFC-phone with server connectivity to purchase services or goods identified by passive tags (smart cards). Unlike the design of reader-based payment protocol, the phone is now responsible for collecting transaction evidence as well as submitting it to a payment server. In this chapter, we will elaborate on the first type of mobile payment protocol and its analyses.

7.2 Overview

The mobile-reader payment protocol, denoted as $\mathcal{O}_{\mathfrak{A}}$, allows an NFC-enabled mobile phone to interact with an NFC reader to make a transaction over NFCIP1 (NFC-LLCP) [NF12b]. We assume the NFC reader is trustworthy (a trustworthy principal is well maintained, honest and runs a correct and authenticated protocol) and it is connected to a payment server via a secure TLS channel [IR08a] [IR08b][1]. The payment server is responsible for collecting transaction evidence and performing operations such as billing and auditing. We will use atomic variables Ph to denote the phone, Re to denote the reader and $\aleph$ to denote a certificate authority that issues and manages certificates.

We let $\Psi(Ph, \delta(ID_{Ph}), \Theta(T_1^{Ph}, T_2^{Ph}), K_{Ph}, \aleph)$ denote the certificate issued to Ph and $\Psi(Re, \delta(ID_{Re}), \Theta(T_1^{Re}, T_2^{Re}), K_{Re}, \aleph)$ denote the certificate issued to Re. In the following chapters, we will use $\Psi(P)$ as a shorthand notation to denote the complete X.509 certificate of a principal P.

The NFC phone does not maintain a state-machine for payment information such as balance. Instead, this information is stored on the payment server. The phone is identified by its certificate and some auxiliary information such as its associated Permanent Account Number (PAN).

There are secure elements [KEAR09] [LKM+05] on the NFC phone and the reader. The embedded secure element stores the private key, as well as any secrets that can be remotely provisioned onto the phone or reader from the operator. Secrets consist of both short-lived symmetric keys as well as protocol programs. It is not possible to read

[1]The communication between NFC reader and the server is assumed to be secure and is outside the scope of this analysis.

or modify the contents outside the secure element, and only authenticated programs can access its corresponding keys or secrets.

We assume there is a single payment server and there are many authenticated readers and phones. Additionally, we assume that one Re can serve only one Ph at anytime, version negotiation and agreement have been successfully carried out, and both Ph and Re have agreed on the communication protocol. These procedures are necessary in real life and are outside the scope of the analyses at our level of abstraction.

7.3 Initial Assumptions Γ^0

Initial assumption Γ^0 is a set of the grounded formulas that are assumed to be valid before a protocol starts. The private keys of Ph and Re are generated inside secure elements. Once generated, they never leave the secure elements, and the corresponding public keys are sent to $\aleph$ for certification. We assume that the certificates have been signed by $\aleph$ and then distributed to Ph and Re prior to the start of the protocol session. Additionally, we assume:

1. Ph and Re know the public key of the certificate authority $\aleph$:

$$Ph \models \wp(\aleph, K_{\aleph}) \tag{7.1}$$

$$Re \models \wp(\aleph, K_{\aleph}) \tag{7.2}$$

2. Both Ph and Re believe that $\aleph$ is trustworthy. Ph believes Re is trustworthy:

$$Ph \models \aleph \mapsto \star \tag{7.3}$$

$$Re \models \aleph \mapsto \star \tag{7.4}$$

$$Ph \models Re \mapsto \star \tag{7.5}$$

3. Ph and Re have access to their own private and public keys:

$$Ph \models \wp(Ph, K_{Ph}) \tag{7.6}$$

$$Re \models \wp(Re, K_{Re}) \tag{7.7}$$

$$Ph \models \psi(Ph, K_{Ph}^{-1}) \tag{7.8}$$

$$Re \models \psi(Re, K_{Re}^{-1}) \tag{7.9}$$

4. ID_{Ph} and ID_{Re} are not on the certificate revocation list and the time stamps on Ph's and Re's certificates are consistent with the current time:

$$Ph \models \chi(\delta(ID_{Re})) \tag{7.10}$$

$$Re \models \chi(\delta(ID_{Ph})) \tag{7.11}$$

$$Ph \models \triangle(\Theta(T_1^{Re}, T_2^{Re})) \tag{7.12}$$

$$Re \models \triangle(\Theta(T_1^{Ph}, T_2^{Ph})) \tag{7.13}$$

5. Re can reconstruct the server-hello message which is to be generated by itself in message 2 in Figure 7-1:

$$Re \models \sqcap (Re, \varkappa(N_{Re}).\Theta(T_{Re}).D_{Tran}) \tag{7.14}$$

6. Ph can reconstruct message D_{Ph}, which is to be generated by itself in message 3 in Figure 7-1:

$$Ph \models \sqcap (Ph, D_{Ph}) \tag{7.15}$$

7. Re and Ph believe the time-stamp $\Theta(T_{Re})$ is fresh:

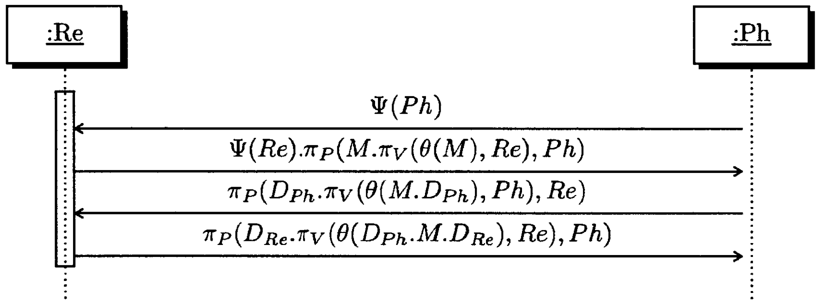
$$Re \models \sharp(\Theta(T_{Re})) \tag{7.16}$$

$$Ph \models \sharp(\Theta(T_{Re})) \tag{7.17}$$

8. Re believes the nonce $\varkappa(N_{Re})$ it generates is fresh:

$$Re \models \sharp(\varkappa(N_{Re})) \tag{7.18}$$

7.4 Protocol $\mathfrak{O}_{\mathfrak{A}}$ Description



:Re
:Ph
$\Psi(Ph)$
$\Psi(Re).\pi_P(M.\pi_V(\theta(M),Re),Ph)$
$\pi_P(D_{Ph}.\pi_V(\theta(M.D_{Ph}),Ph),Re)$
$\pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}),Re),Ph)$

Figure 7-1: UML sequence diagram for the NFC phone-reader payment protocol $\mathfrak{O}_{\mathfrak{A}}$.

The UML sequence diagram of the protocol $\mathfrak{O}_{\mathfrak{A}}$ is shown in Figure 7-1. Below is a step-by-step description of the protocol.

7.4.1 Pre-Conditions and Transmissions

PreCond Step 1. *Nil.*

$$\Omega_1 = \emptyset \tag{7.19}$$

Step 1 (Client Hello). *Ph first sends its certificate as a client-hello to Re.*

$$Ph \rightarrow Re : \Psi(Ph) \tag{7.20}$$

PreCond Step 2. *Re validates Ph's certificate. Instrumental in the process is the enforcement of certificate expiration and revocation. The validation must be successful* *and thus Re can get the public key of Ph:*

$$\Omega_2 = \{Re \models \wp(Ph, K_{Ph})\} \tag{7.21}$$

Step 2 (Server Hello). *Upon receiving client-hello from Ph, Re constructs a challenge M consisting of a fresh nonce $\varkappa(N_{Re})$, Re's time-stamp T_{Re}, a description of the transaction D_{Tran} which may include financial information on the amount charged, merchant, location etc, and the recipient identity $\delta(ID_{Ph})$, which Re gets from the $\Psi(Ph)$ received in the previous step. M is formalized in $TGPay$ as:*

$$M = \varkappa(N_{Re}).\Theta(T_{Re}).D_{Tran}.\delta(ID_{Ph}) \tag{7.22}$$

Re has a privacy requirement over M, since D_{Tran} contains transactional information that cannot be revealed. Re then sends its certificate and a cipher-text encrypted using Ph's public key to Ph:

$$Re \rightarrow Ph : \Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph) \tag{7.23}$$

PreCond Step 3. *Ph needs to validate Re's certificate. It also checks for integrity, freshness and if it is the intended recipient (if its ID is acknowledged). It may also perform some semantic validations that are transaction protocol specific. For example, it checks if the transaction request packet D_{Tran} is correct and as expected. Since these are not closely related to the security model, we will omit them from the pre-conditions. However, they can be easily added. In summary, the pre-conditions for this transmission are:*

1. *Ph believes Re's public key:*

$$Ph \models \wp(Re, K_{Re}) \tag{7.24}$$

2. Ph believes M comes from Re:

$$Ph \models \mathfrak{S}(M, Re) \tag{7.25}$$

3. Ph believes M is fresh:

$$Ph \models \sharp(M) \tag{7.26}$$

4. Ph believes M is intact since uttered by Re:

$$Ph \models \eta(M, Re) \tag{7.27}$$

5. Ph believes that it is the intended reader for M and that M acknowledges its ID:

$$Ph \models \mathfrak{R}(M, Ph) \tag{7.28}$$

$$Ph \models \varpi(M, \delta(ID_{Ph})) \tag{7.29}$$

where for 7.29 $Ph \models \varpi(M, \delta(ID_{Ph}))$, $\delta(ID_{Ph})$ is the identity of Ph contained in $\Psi(Ph)$. This acknowledgement assertion models a check that Ph performs. If the identity acknowledged by Re is a different one from $\delta(ID_{Ph})$, then Ph knows something is wrong.

Step 3 (Phone Binding). *Ph next constructs its description message D_{Ph}. D_{Ph} will contain any additional information that Ph needs to provide to complete this transaction, such as the CHD. Ph next produces signature S_{Ph}. The signature binds Ph's belief over $M.D_{Ph}$.*

$$S_{Ph} = \pi_V(\theta(M.D_{Ph}), Ph) \tag{7.30}$$

After that, Ph encrypts D_{Ph} and the signature using Re's public key and sends it to Re.

$$Ph \rightarrow Re : \pi_P(D_{Ph}.S_{Ph}, Re) \tag{7.31}$$

PreCond Step 4. *Re checks the integrity, freshness and authenticity of the message. Additionally, Re checks to see if Ph is acknowledging the correct M (the M constructed by Re, not the one sent by Ph). Together, the pre-conditions at this step are:*

1. *Re believes that Ph sent its affirmation of the transaction, represented by $M.D_{Ph}$:*

$$Re \models \mathfrak{S}(M.D_{Ph}, Ph) \tag{7.32}$$

2. *Re believes that $M.D_{Ph}$ is freshly generated not a replay:*

$$Re \models \sharp(M.D_{Ph}) \tag{7.33}$$

3. *Re believes that $M.D_{Ph}$ is intact since it was uttered by Ph:*

$$Re \models \eta(M.D_{Ph}, Ph) \tag{7.34}$$

Step 4 (Receipt). *Upon receiving D_{Ph} and after validating its integrity and authenticity, Re generates a receipt and transmits it to Ph. The receipt, denoted as D_{Re}, is a composition of the transaction request D_{Tran} and payer's response D_{Ph} as well as the reader's recognition of the binding of the two. Let R_{Re}^{Ph} denote the receipt generated by Re to be sent to Ph:*

$$R_{Re}^{Ph} = D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re) \tag{7.35}$$

Finally, R_{Re}^{Ph} is encrypted and sent to Ph. Thus,

$$Re \rightarrow Ph : \pi_P(R_{Re}^{Ph}, Ph) \tag{7.36}$$

7.4.2 Post-Conditions

PostCond 1. *The set of post-conditions Σ we impose on this protocol consists of $\Sigma_{(+)}$, Ph's assertions on the last message it receives; and, $\Sigma_{(-)}$, assertions from all principals on what the protocol should achieve after it completes successfully.*

$\Sigma_{(+)}$: *Ph checks to see if Re was acknowledging the correct D_{Ph} (Ph recently sent D_{Ph}) and that $D_{Ph}.M.D_{Re}$ is from Re and is fresh and intact.*

$$Ph \models \mathfrak{S}(D_{Ph}.M.D_{Re}, Re) \tag{7.37}$$

$$Ph \models \sharp(D_{Ph}.M.D_{Re}) \tag{7.38}$$

$$Ph \models \eta(D_{Ph}.M.D_{Re}, Re) \tag{7.39}$$

$\Sigma_{(-)}$: *(a) Irrefutable request from Re: Ph holds non-repudiatable and intact evidence that Re has requested payment (of particular amount at particular time, at particular location, etc.) If Re refutes, Ph is able to bring forward a transferable proof to an arbitrator:*

$$Ph \models \sqsupset(Ph, \mathfrak{S}(M, Re)) \tag{7.40}$$

$$Ph \models \sqsupset(Ph, \eta(M, Re)) \tag{7.41}$$

where the first formula provides non-repudiation on content and the second formula provides non-repudiation on integrity of the content.

(b) Irrefutable payment from Ph: Ph has authorized its payment in response to Re's request. Ph provided information such as PAN number and the amount it authorized to pay, as well as its willingness to associate this payment with Re's request. Re holds this non-repudiatable evidence. If Ph refutes any details on the amount, PAN, merchant, payment time and location, Re can

bring forward a transferable proof:

$$Re \models \beth(Re, \mathfrak{S}(M.D_{Ph}, Ph)) \tag{7.42}$$

$$Re \models \beth(Re, \eta(M.D_{Ph}, Ph)) \tag{7.43}$$

(c) *Irrefutable payment confirmation: Additionally, Re cannot refute later that it has not received an acceptable payment from Ph. Ph can bring forward a transferable proof to an arbitrator about Re's acceptance of Ph's payment:*

$$Ph \models \beth(Ph, \mathfrak{S}(D_{Ph}.M.D_{Re}, Re)) \tag{7.44}$$

$$Ph \models \beth(Ph, \eta(D_{Ph}.M.D_{Re}, Re)) \tag{7.45}$$

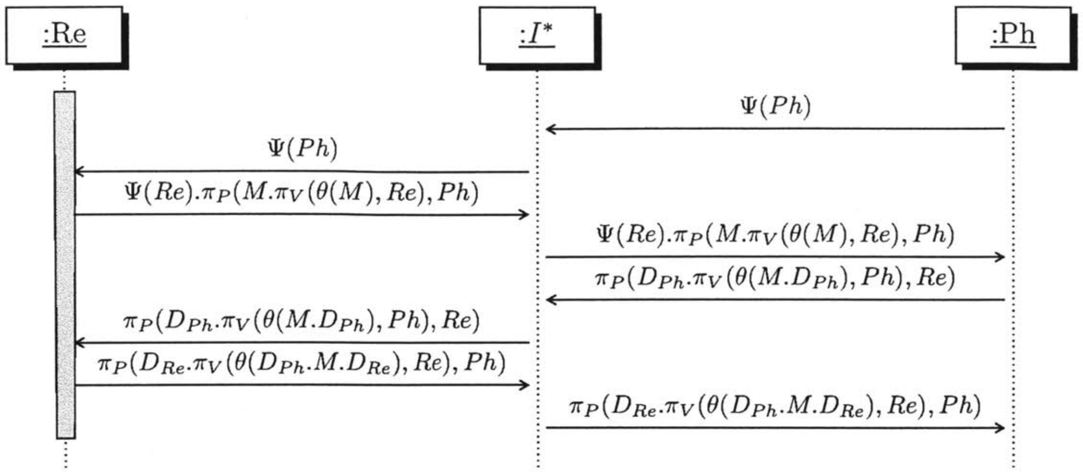
7.5 PTGPA-Alpha Analysis of $\mathfrak{O}_{\mathfrak{A}}$

We ran the PTGPA-Alpha algorithm. The algorithm found the protocol ran to completion successfully. All post-conditions are met at completion. Thus we conclude that $\mathfrak{O}_{\mathfrak{A}}$ is Alpha-S correct by Theorem 9. This means that the candidate protocol $\mathfrak{O}_{\mathfrak{A}}$ is consistent with the set of security requirements as reflected in the specified pre- and post-conditions.

In the remainder of this chapter, we specify two attacks obtained by applying transformation function $\mathbb{A}$ in Definition 37 on $\mathfrak{O}_{\mathfrak{A}}$. We then discuss how these attacks are defeated by $\mathfrak{O}_{\mathfrak{A}}$ using the formal PTGPA-Beta analysis framework.

7.6 Secrecy Attack

In this section, we introduce an eavesdropping intruder that silently listens to all transmissions between Re and Ph without making any modifications to the messages exchanged. This attack over $\mathfrak{O}_{\mathfrak{A}}$ is denoted as $\mathfrak{O}^*_{\mathfrak{A}}$. Our goal is to verify that the secrecy requirements of sensitive information are satisfied. The sequence diagram of $\mathfrak{O}^*_{\mathfrak{A}}$ is presented in Figure 7-2.



:Re
:I^*
:Ph
$\Psi(Ph)$
$\Psi(Ph)$
$\Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph)$
$\Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph)$
$\pi_P(D_{Ph}.\pi_V(\theta(M.D_{Ph}), Ph), Re)$
$\pi_P(D_{Ph}.\pi_V(\theta(M.D_{Ph}), Ph), Re)$
$\pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re), Ph)$
$\pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re), Ph)$

Figure 7-2: UML sequence diagram for the attack protocol $\mathfrak{O}^*_{\mathfrak{A}}$.

7.6.1 Attack $\mathfrak{O}^*_{\mathfrak{A}}$ at a Glance

The set of initial assumptions is the same as that of $\mathfrak{O}_{\mathfrak{A}}$, except that the intruder, denoted as I^*, knows the public keys of certificate authority $\aleph$, Re and Ph. Thus, we have $I^* \models \wp(\aleph, K_\aleph)$, $I^* \models \wp(Ph, K_{Ph})$, $I^* \models \wp(Re, K_{Re})$. Additionally, we assume $\aleph$ and Re are trustworthy to the intruder: $(I^* \models \aleph \mapsto \star,\ I^* \models Re \mapsto \star)$

The post-conditions Σ^* are the same as Σ of $\mathfrak{O}_{\mathfrak{A}}$, since I^* does not modify any transmissions, and all principals from $\mathfrak{O}_{\mathfrak{A}}$ fully participate in $\mathfrak{O}^*_{\mathfrak{A}}$. The sequence of transmissions, and pre-conditions are presented in Figure 7-3.

The set of intruder assertions models secrecy using negations of 3 formulas. They assert the intruder's beliefs on not being able to reconstruct M, D_{Re} or D_{Ph}:

1. Intruder I^* cannot reconstruct transactional request message M (Secrecy on M)

$$\neg I^* \models \sqcap (I^*, M) \tag{7.46}$$

2. Intruder I^* cannot reconstruct transactional affirmation message D_{Ph} (Secrecy on D_{Ph})

$$\neg I^* \models \sqcap (I^*, D_{Ph}) \tag{7.47}$$

Seq.	Pre-Conditions	Transmission
1	$\emptyset$	$Ph \rightarrow I^* : \Psi(Ph)$
2	$\emptyset$	$I^* \rightarrow Re : \Psi(Ph)$
3	$Re \models \wp(Ph, K_{Ph})$	$Re \rightarrow I^* : \Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph)$
4	$\emptyset$	$I^* \rightarrow Ph : \Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph)$
5	$Ph \models \wp(Re, K_{Re})$, $Ph \models \mathfrak{S}(M, Re)$, $Ph \models \sharp(M)$, $Ph \models \eta(M, Re)$, $Ph \models \mathfrak{R}(M, Ph)$, $Ph \models \varpi(M, \delta(ID_{Ph}))$	$Ph \rightarrow I^* : \pi_P(D_{Ph}.\pi_V(\theta(M.D_{Ph}), Ph), Re)$
6	$\emptyset$	$I^* \rightarrow Re : \pi_P(D_{Ph}.\pi_V(\theta(M.D_{Ph}), Ph), Re)$
7	$Re \models \mathfrak{S}(M.D_{Ph}, Ph)$, $Re \models \sharp(M.D_{Ph})$, $Re \models \eta(M.D_{Ph}, Ph)$	$Re \rightarrow I^* : \pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re), Ph)$
8	$\emptyset$	$I^* \rightarrow Ph : \pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re), Ph)$

Figure 7-3: Pre-conditions and message exchanges in attack $\mathfrak{O}^*_{\mathfrak{A}}$

3. Intruder I^* cannot reconstruct message D_{Re} (Secrecy on D_{Re})

$$\neg I^* \models \sqcap (I^*, D_{Re}) \tag{7.48}$$

7.6.2 PTGPA-Beta Analysis of $\mathfrak{O}^*_{\mathfrak{A}}$

We ran the PTGPA-Beta algorithm against $\mathfrak{O}^*_{\mathfrak{A}}$. The protocol ran to the end and all post-conditions and intruder assertions were valid. The algorithm returns true and thus $\mathfrak{O}_{\mathfrak{A}}$ is correct under attack $\mathfrak{O}^*_{\mathfrak{A}}$ by Theorem 10.

In $\mathfrak{O}^*_{\mathfrak{A}}$, we inserted an intruder that listened to all messages exchanged between Re and Ph. The proof system showed that the intruder was unable to uncover plain-text M, D_{Re} or D_{Ph}. The attack was not detected and all post-conditions and intruder assertions were met. This means that the original protocol is secure against this specific type of eavesdropping attack for the secrecy of the transactional information exchanged.

7.7 Replay Attack

We now model another attack, in which an intruder I^* attempts to impersonate Ph by replaying its certificate and some of the old messages I^* eavesdropped on in a previous communication that is similar to $\mathfrak{O}^*_{\mathfrak{A}}$. We denote the attack as $\mathfrak{O}^{\#}_{\mathfrak{A}}$; the UML sequence of message transmissions is presented in Figure 7-4:

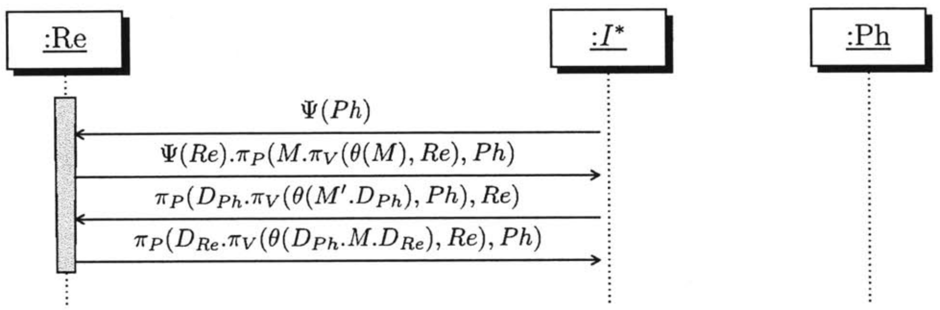

:Re
$:I^*$
:Ph
$\Psi(Ph)$
$\Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph)$
$\pi_P(D_{Ph}.\pi_V(\theta(M'.D_{Ph}), Ph), Re)$
$\pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re), Ph)$

Figure 7-4: UML sequence diagram for the attack protocol $\mathfrak{O}^{\#}_{\mathfrak{A}}$.

7.7.1 Attack $\mathfrak{O}^{\#}_{\mathfrak{A}}$ at a Glance

In this protocol, Re thinks it is talking to Ph. Instead, Re is interacting with an intruder I^* and Ph is blocked from receiving any messages. After receiving client-hello, a replay of Ph's certificate from I^*, Re returns its certificate as well as $\pi_P(M.\pi_V(\theta(M), Re), Ph)$. Since I^* does not know the private key of Ph, I^* cannot manipulate this message, nor can I^* see its content. Instead, I^* sends a replay of the third transmission in a previous run of $\mathfrak{O}_{\mathfrak{A}}$ between Re and Ph, denoted as $\pi_P(D_{Ph}.\pi_V(\theta(M'.D_{Ph}), Ph), Re)$, where M' is different from M due to the lack of freshness of its construction, and D_{Ph} was the payment information Ph used, in hoping that Re will accept the payment information. The protocol finishes, if it does, with Re sending its signature over $D_{Ph}.M.D_{Re}$ where D_{Ph} was received by Re; M and D_{Re} were constructed by Re.

The initial assumptions are same as that of $\mathfrak{O}^*_{\mathfrak{A}}$. The pre-conditions and message transmissions are presented in Figure 7-5.

Seq.	Pre-Conditions	Transmission
1	$\emptyset$	$I^* \rightarrow Re : \Psi(Ph)$
2	$Re \models \wp(Ph, K_{Ph})$	$Re \rightarrow I^* : \Psi(Re).\pi_P(M.\pi_V(\theta(M), Re), Ph)$
3	$\emptyset$	$I^* \rightarrow Re : \pi_P(D_{Ph}\pi_V(\theta(M'.D_{Ph}), Ph), Re)$
4	$Re \models \mathfrak{S}(M.D_{Ph}, Ph)$, $Re \models \sharp(M.D_{Ph})$, $Re \models \eta(M.D_{Ph}, Ph)$	$Re \rightarrow I^* : \pi_P(D_{Re}.\pi_V(\theta(D_{Ph}.M.D_{Re}), Re), Ph)$

Figure 7-5: Protocol $\mathfrak{O}^{\#}_{\mathfrak{A}}$ for mobile-reader payment protocol

We see all pre-conditions related to Ph are removed since Ph did not participate in this attack protocol. All pre-conditions for message transmissions in which the intruder is the sender are empty sets. Moreover, the three pre-conditions for message 4 are assertions on M not M'. This is because these assertions are Ph's acknowledgements of M constructed by Re, not the M Re receives from Ph. They assert that, at this step of the protocol, Re must be able to derive the belief that M, as constructed by Re, is acknowledged by Ph. Ph acknowledging a different M', as is the case here, does not alter the acknowledgement target that Re is seeking.

The post-conditions $\Sigma_{(+)}$ are now the empty set and $\Sigma_{(-)}$ contains only the assertions from Re since Ph does not participate fully in $\mathfrak{O}^{\#}_{\mathfrak{A}}$. The post-conditions $\Sigma_{(-)}$ are:

1. Re's belief of the non-repudiatable evidence of Ph's transaction affirmation $M.D_{Ph}$

$$Re \models \sqsupset(Re, \mathfrak{S}(M.D_{Ph}, Ph)) \tag{7.49}$$

2. Re's belief of the integrity of the non-repudiatable evidence

$$Re \models \sqsupset(Re, \eta(M.D_{Ph}, Ph)) \tag{7.50}$$

Finally, the intruder assertions of this attack protocol are:

1. Intruder I^* cannot reconstruct transactional request message M (Secrecy on M)

$$\neg I^* \models \sqcap (I^*, M) \tag{7.51}$$

2. Intruder I^* cannot reconstruct D_{Re}, the receipt

$$\neg I^* \models \sqcap (I^*, D_{Re}) \tag{7.52}$$

They reflect the secrecy requirements we imposed on the payment information M and D_{Re}; both were constructed and sent by the honest principal Re.

7.7.2 PTGPA-Beta Analysis of $\mathfrak{O}^{\#}_{\mathfrak{A}}$

We ran the PTGPA-Beta algorithm against $\mathfrak{O}^{\#}_{\mathfrak{A}}$. The algorithm found the protocol halted after Re received $\pi_P(D_{Ph}.\pi_V(\theta(M'.D_{Ph}), Ph), Re)$ due to a violation of pre-condition $Re \models \mathfrak{S}(M.D_{Ph}, Ph)$, where M was the challenge Re freshly generated at the beginning of the protocol. Moreover, at the time of violation, Υ was valid. That is, the intruder cannot reconstruct M or D_{Re}. This means that the original protocol can detect this type of attack and at the time of detection, no secrecy requirements have been violated.

The PTGPA-Beta algorithm thus returns true, and we conclude that $\mathfrak{O}_{\mathfrak{A}}$ is correct under impersonation attack $\mathfrak{O}^{\#}_{\mathfrak{A}}$ by Theorem 10.

7.8 Summary

This chapter introduced a high security contactless mobile-reader payment protocol. Two key features in this protocol that are not found in the EMV DDA are the secrecy on the payment information (analogous to the CHD in EMV) and the handling of non-repudiation for both merchant and customer. We formalized the candidate protocol in *TGPay* and we showed that it was Alpha-S correct. We also discussed two forms of attack, one on secrecy ($\mathfrak{O}^*_{\mathfrak{A}}$) and one on impersonation ($\mathfrak{O}^{\#}_{\mathfrak{A}}$). We conclude that the candidate protocol $\mathfrak{O}_{\mathfrak{A}}$ is secure against these two attacks, with respect to the set of initial assumptions and pre- and post-conditions that we specified.

Chapter 8

Mobile-Tag Payment Protocols

Contents

8.1 Introduction

This section presents a mobile payment protocol that does not use an NFC-reader. The protocol, denoted as $\mathfrak{D}_{\mathfrak{B}}$, allows an NFC-phone with server data connectivity to purchase services or goods identified by passive tags (smart cards). Unlike the design of the reader-based payment protocol, the phone is now responsible for collecting transaction evidence as well as submitting it to a payment server. The passive tags are placed with descriptions of the services or goods for purchase. They are unpowered and they communicate with the phone in a load modulation scheme. Popular examples are smart posters, where passive tags for the advertisement of services are mounted on the wall, as well as smart displays in a shopping mall, where customers can tap to pay for goods displayed. Customers can then pick up the goods purchased when they exit the store or mall.

The main difference between this type of payment protocol and the reader-based protocol is that passive tags cannot represent arbitrary services or goods. Instead, they represent a fixed set of services or goods. The advantage is that there is often no privacy requirement imposed on the items for sale - everyone is welcome to check them out. After a phone taps on a passive tag, the tag presents a "menu" of goods or service for purchase. The same "menu" is returned upon every such inquiry. The only requirement the payment server mandates is that an identified phone (optionally with selected payment instrument information) has agreed to purchase an identified service at a specific time. In this chapter, we will present a generic phone-tag payment protocol and formalize it in *TGPay*.

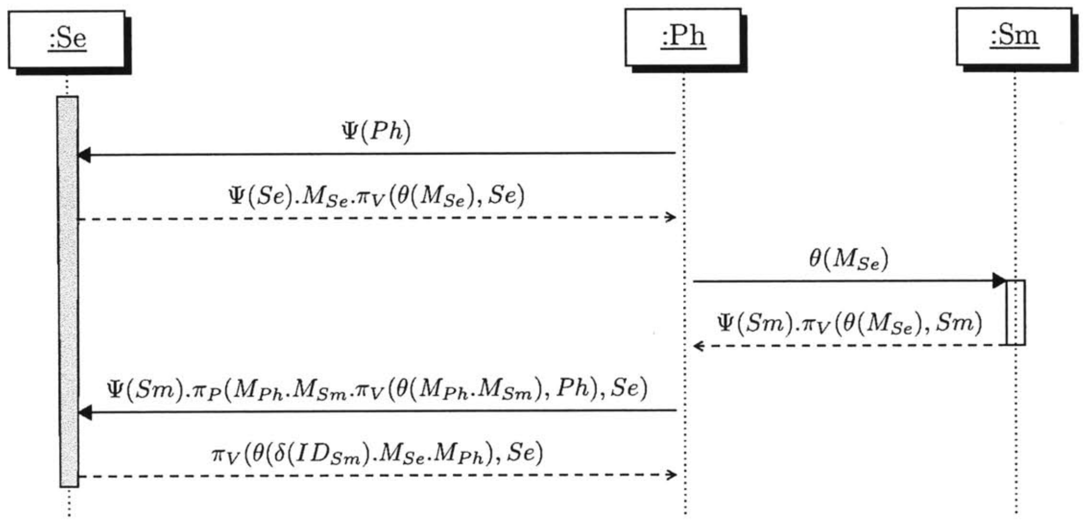
8.2 Overview

We assume the passive tags can perform primitive cryptographic operations such as encryption and digital signature. Additionally, the tags have sufficient memory to store their certificates. Storage of private keys and secrets in these tags are assumed to be analogous to the secure element used in a phone. Private keys and any secrets cannot be read or modified by intruders.

There are three types of principals involved in this protocol: a unique payment server Se, a number of valid NFC-phones Ph, and a number of valid smart cards Sm. We assume that Se is considered trustworthy by both Ph and Sm. Ph and Sm are not considered trustworthy by Se. We will assume an issuing authority $\aleph$ that manages the certificates for all principals.

A high-level description of the protocol is as follows: After Se receives the client-hello from a Ph, Se generates a challenge $M_{Se} = \varkappa(N_{Se}).\Theta(T_{Se}).\delta(ID_{Ph})$, where $\varkappa(N_{Se})$

is a globally unique nonce and $\Theta(T_{Se})$ is a server generated time stamp. Message M_{Se} and its proof of origin is then sent to Ph. Ph next forwards the same message to Sm to sign. After receiving the signature from Sm, Ph performs a second signing over the signature returned from Sm. The resulting doubly-signed signature is transmitted back to Se for validation. Finally, if everything is correct, a receipt is returned to Ph as proof. The overall sequence of message flow is presented in Figure 8-1.



:Se
:Ph
:Sm
$\Psi(Ph)$
$\Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
$\theta(M_{Se})$
$\Psi(Sm).\pi_V(\theta(M_{Se}), Sm)$
$\Psi(Sm).\pi_P(M_{Ph}.M_{Sm}.\pi_V(\theta(M_{Ph}.M_{Sm}), Ph), Se)$
$\pi_V(\theta(\delta(ID_{Sm}).M_{Se}.M_{Ph}), Se)$

Figure 8-1: UML sequence diagram for the NFC phone - Smart Card payment protocol $\mathfrak{O}_{\mathfrak{B}}$.

We now present the formalized specification of $\mathfrak{O}_{\mathfrak{B}}$, including initial assumptions, message transmissions, pre-conditions as well as post-conditions.

8.3 Initial Assumptions Γ^0

1. Both Ph and Sm believe that Se is trustworthy. Ph, Sm and Se believe that $\aleph$ is trustworthy. Thus:

$$Ph \models Se \mapsto \star \tag{8.1}$$

$$Sm \models Se \mapsto \star \tag{8.2}$$

$$Ph \models \aleph \mapsto \star \tag{8.3}$$

$$Sm \models \aleph \mapsto \star \tag{8.4}$$

$$Se \models \aleph \mapsto \star \tag{8.5}$$

2. Ph, Sm and Se have access to their own private and public keys:

$$Ph \models \wp(Ph, K_{Ph}) \tag{8.6}$$

$$Sm \models \wp(Sm, K_{Sm}) \tag{8.7}$$

$$Se \models \wp(Se, K_{Se}) \tag{8.8}$$

$$Ph \models \psi(Ph, K_{Ph}^{-1}) \tag{8.9}$$

$$Sm \models \psi(Sm, K_{Sm}^{-1}) \tag{8.10}$$

$$Se \models \psi(Se, K_{Se}^{-1}) \tag{8.11}$$

3. Ph, Sm and Se have root certificates from the certificate authority $\aleph$ and hence know $\aleph$'s public key:

$$Ph \models \wp(\aleph, K_{\aleph}) \tag{8.12}$$

$$Sm \models \wp(\aleph, K_{\aleph}) \tag{8.13}$$

$$Se \models \wp(\aleph, K_{\aleph}) \tag{8.14}$$

4. ID_{Ph}, ID_{Se}, ID_{Sm} are not on the certificate revocation list and the time stamps

on Ph's, Se's and Sm's certificates are consistent with current time:

$$Ph \models \chi(\delta(ID_{Se})) \tag{8.15}$$

$$Sm \models \chi(\delta(ID_{Se})) \tag{8.16}$$

$$Se \models \chi(\delta(ID_{Se})) \tag{8.17}$$

$$Se \models \chi(\delta(ID_{Ph})) \tag{8.18}$$

$$Sm \models \chi(\delta(ID_{Ph})) \tag{8.19}$$

$$Ph \models \chi(\delta(ID_{Ph})) \tag{8.20}$$

$$Se \models \chi(\delta(ID_{Sm}))) \tag{8.21}$$

$$Ph \models \chi(\delta(ID_{Sm})) \tag{8.22}$$

$$Sm \models \chi(\delta(ID_{Sm})) \tag{8.23}$$

$$Ph \models \triangle(\Theta(T_1^{Se}, T_2^{Se})) \tag{8.24}$$

$$Sm \models \triangle(\Theta(T_1^{Se}, T_2^{Se})) \tag{8.25}$$

$$Se \models \triangle(\Theta(T_1^{Se}, T_2^{Se})) \tag{8.26}$$

$$Se \models \triangle(\Theta(T_1^{Ph}, T_2^{Ph})) \tag{8.27}$$

$$Sm \models \triangle(\Theta(T_1^{Ph}, T_2^{Ph})) \tag{8.28}$$

$$Ph \models \triangle(\Theta(T_1^{Ph}, T_2^{Ph})) \tag{8.29}$$

$$Ph \models \triangle(\Theta(T_1^{Sm}, T_2^{Sm})) \tag{8.30}$$

$$Se \models \triangle(\Theta(T_1^{Sm}, T_2^{Sm})) \tag{8.31}$$

$$Sm \models \triangle(\Theta(T_1^{Sm}, T_2^{Sm})) \tag{8.32}$$

5. Se can reconstruct the nonce and time-stamp generated by itself, $\varkappa(N_{Se}).\Theta(T_{Se})$; Ph can reconstruct message M_{Ph} generated by itself:

$$Se \models \sqcap (Se, \varkappa(N_{Se}).\Theta(T_{Se})) \tag{8.33}$$

$$Ph \models \sqcap (Ph, M_{Ph}) \tag{8.34}$$

6. Se and Ph believe that $\Theta(T_{Se})$, generated by Se, is a fresh time-stamp:

$$Se \models \triangle(\Theta(T_{Se})) \tag{8.35}$$

$$Ph \models \triangle(\Theta(T_{Se})) \tag{8.36}$$

7. Se believes the nonce $\varkappa(N_{Se})$ is fresh:

$$Se \models \sharp(\varkappa(N_{Se})) \tag{8.37}$$

8.4 Protocol $\mathfrak{D}_{\mathfrak{B}}$ Description

8.4.1 Pre-conditions and Transmissions

PreCond Step 1.

$$\Omega_1 = \emptyset \tag{8.38}$$

Step 1. *Ph sends its certificate to Se.*

$$Ph \rightarrow Se : \Psi(Ph) \tag{8.39}$$

PreCond Step 2. *Se needs to verify the certificate of Ph and thereby obtain the public key of Ph.*

$$Se \models \wp(Ph, K_{Ph}) \tag{8.40}$$

Step 2. *Se generates M_{Se} which consists of a nonce $\varkappa(N_{Se})$, server time-stamp $\Theta(T_{Se})$ as well as a recipient tag $\delta(ID_{Ph})$ that Se gets from $\Psi(Ph)$ it received in the previous message. Se has no privacy requirement over M_{Se}. It then sends its certificate $\Psi(Se)$, M_{Se} and its signature over the hash of M_{Se} to Ph. We assume that Se keeps track of* *all M_{Se} it constructs for a specific Ph.*

$$Se \rightarrow Ph : \Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se) \tag{8.41}$$

PreCond Step 3. *The pre-conditions at this step are:*

1. *Ph needs to extract the public key of Se and ensure that M_{Se} comes from Se:*

$$Ph \models \wp(Se, K_{Se}) \tag{8.42}$$

$$Ph \models \mathfrak{S}(M_{Se}, Se) \tag{8.43}$$

2. *Ph needs to ensure that M_{Se} is fresh and intact:*

$$Ph \models \sharp(M_{Se}) \tag{8.44}$$

$$Ph \models \eta(M_{Se}, Se) \tag{8.45}$$

3. *Ph needs to ensure that M_{Se} acknowledges Ph's ID, so that it is intended for Ph and not any other Ph':*

$$Ph \models \varpi(M_{Se}, \delta(ID_{Ph})) \tag{8.46}$$

Step 3. *Ph next sends hashed M_{Se} to Sm:*

$$Ph \rightarrow Sm : \theta(M_{Se}) \tag{8.47}$$

PreCond Step 4.

$$\Omega_4 = \emptyset \tag{8.48}$$

Step 4. *Sm computes its signature over the hash of M_{Se} and returns the signature as* *well as its certificate.*

$$Sm \rightarrow Ph : \Psi(Sm).\pi_V(\theta(M_{Se}), Sm) \tag{8.49}$$

PreCond Step 5. *The pre-conditions at this step are:*

1. *Ph needs to first verify the certificate of Sm (validity, expiration, CRL etc.) and extract the public key of Sm:*

$$Ph \models \wp(Sm, K_{Sm}) \tag{8.50}$$

2. *Ph needs to check that Sm acknowledged (uttered) M_{Se} to rule out any replay attack. Ph also checks if M_{Se} is intact:*

$$Ph \models \mathfrak{S}(M_{Se}, Sm) \tag{8.51}$$

$$Ph \models \eta(M_{Se}, Sm) \tag{8.52}$$

Step 5. *Let $M_{Sm} = \pi_V(\theta(M_{Se}), Sm)$; M_{Sm} can be seen as a freshly generated and globally unique payment request, in which the service and merchant are identified through Sm, and the time and buyer are identified by M_{Se}. Ph next prepares its payment details, such as a selected PAN number, denoted as a string M_{Ph}. Ph signs the hash of $M_{Ph}.M_{Sm}$ and thereby effectively binds its agreement over M_{Ph} and M_{Sm}. Finally, Ph transmits Sm's certificate, an encrypted string over $M_{Ph}.M_{Sm}$, and the signed hash.*

$$Ph \rightarrow Se : \Psi(Sm).\pi_P(M_{Ph}.M_{Sm}.\pi_V(\theta(M_{Ph}.M_{Sm}), Ph), Se) \tag{8.53}$$

PreCond Step 6. *The server needs to verify a number of items before sending out the receipt.*

1. *First it verifies the certificate of Sm, $\Psi(Sm)$, and extracts Sm's public key:*

$$Se \models \wp(Sm, K_{Sm}) \tag{8.54}$$

2. *Se then retrieves M_{Se} that was sent to Ph from its database. Se verifies that M_{Se} is acknowledged by Sm and the acknowledgement is intact:*

$$Se \models \mathfrak{S}(M_{Se}, Sm) \tag{8.55}$$

$$Se \models \eta(M_{Se}, Sm) \tag{8.56}$$

3. *Se needs to verify that $M_{Ph}.M_{Sm}$ is uttered by Ph and is intact:*

$$Se \models \mathfrak{S}(M_{Ph}.M_{Sm}, Ph) \tag{8.57}$$

$$Se \models \eta(M_{Ph}.M_{Sm}, Ph) \tag{8.58}$$

Step 6. *Se next issues a receipt. It produces a signature over the hash of $\delta(ID_{Sm})$, M_{Se} and M_{Ph}, which effectively acknowledges the transaction between $\delta(ID_{Ph})$ and $\delta(ID_{Sm})$, at the time $\Theta(T_{Se})$ with payment information as described in M_{Ph}:*

$$Se \rightarrow Ph : \pi_V(\theta(\delta(ID_{Sm}).M_{Se}.M_{Ph}), Se) \tag{8.59}$$

8.4.2 Post-conditions

PostCond 1. *The set of post-conditions Σ we impose on this protocol consists of the following two components, $\Sigma_{(+)}$ and $\Sigma_{(-)}$:*

$\Sigma_{(+)}$ *: Ph's assertions on the last message that Ph received. Ph asserts that it comes from Se, it is intact, and infers $\delta(ID_{Sm}).M_{Se}.M_{Ph}$.*

$$\Sigma_{(+)} = \{Ph \models \mathfrak{S}(\delta(ID_{Sm}).M_{Se}.M_{Ph}, Se) \tag{8.60}$$

$$Ph \models \eta(\delta(ID_{Sm}).M_{Se}.M_{Ph}, Se)\} \tag{8.61}$$

$\Sigma_{(-)}$ *: Assertions on the overall protocol at completion.*

(a) Irrefutable payment from Ph: Ph provided its payment info M_{Ph} and has bound itself to the service and time. If Ph later refutes that it has agreed to

pay for services as identified by smart card identity $\delta(ID_{Sm})$ at time $\Theta(T_{Se})$ with payment details M_{Ph}, the payment server Se can bring forward non-repudiatable evidence to an arbitrator. Formally, this is:

$$Se \models \beth(Se, \mathfrak{S}(M_{Se}, Sm)) \tag{8.62}$$

$$Se \models \beth(Se, \eta(M_{Se}, Sm)) \tag{8.63}$$

$$Se \models \beth(Se, \mathfrak{S}(M_{Ph}.M_{Sm}, Ph)) \tag{8.64}$$

$$Se \models \beth(Se, \eta(M_{Ph}.M_{Sm}, Ph)) \tag{8.65}$$

Through the evidence of M_{Sm} and string M_{Se}, Se offers an transferable proof that Sm bound its identity to M_{Se} (8.62 and 8.63). Moreover, Se offers an transferable proof that Ph bound its identity to payment info M_{Ph} and the same M_{Sm} (8.64 and 8.65), and therefore proves that Ph agreed to purchase the service as identified by $\delta(ID_{Sm})$ with payment info M_{Ph} and at a specific time as identified from $\Theta(T_{Se})$.

(b) *Irrefutable Payment Confirmation: Additionally, Se cannot refute later that it has not received an acceptable payment from Ph. Ph can bring forward a transferable proof to an arbitrator about Se's acceptance of Ph's payment. The evidence binds Se's acceptance to the identities of Ph and Sm, as well as the payment information and time of transaction.*

$$Ph \models \beth(Ph, \mathfrak{S}(\delta(ID_{Sm}).M_{Se}.M_{Ph}, Se)) \tag{8.66}$$

$$Ph \models \beth(Ph, \eta(\delta(ID_{Sm}).M_{Se}.M_{Ph}, Se)) \tag{8.67}$$

8.5 PTGPA-Alpha Analysis of $\mathfrak{O}_{\mathfrak{B}}$

We ran the PTGPA-Alpha algorithm against $\mathfrak{O}_{\mathfrak{B}}$. $\mathfrak{O}_{\mathfrak{B}}$ successfully ran to the end without violating any pre-conditions. Additionally, all post-conditions were valid at the completion. The PTGPA-Alpha algorithm returns true and thus $\mathfrak{O}_{\mathfrak{B}}$ is Alpha-S correct by Theorem 9. This means that the candidate protocol $\mathfrak{O}_{\mathfrak{B}}$ is consistent with the set of business requirements and security goals as reflected in the pre- and post-conditions. Additionally, it is consistent at the prerequisites of the aforementioned set of initial assumptions. All initial assumptions must be first met before one can claim the correctness.

We now consider two potential attacks obtained by applying transformation function $\mathbb{A}$ in Definition 37 to $\mathfrak{O}_{\mathfrak{B}}$.

8.6 Secrecy Attack

In this section, we introduce an intruder that silently listens to all transmission traffic without inducing any modifications to the transmissions. This is an eavesdropping attack over $\mathfrak{O}_{\mathfrak{B}}$, denoted as $\mathfrak{O}^*_{\mathfrak{B}}$. The sequence diagram of $\mathfrak{O}^*_{\mathfrak{B}}$ is presented in Figure 8-2.

The protocol is similar to $\mathfrak{O}_{\mathfrak{B}}$ except that every transmission in $\mathfrak{O}_{\mathfrak{B}}$ is now passed through the intruder I^*.

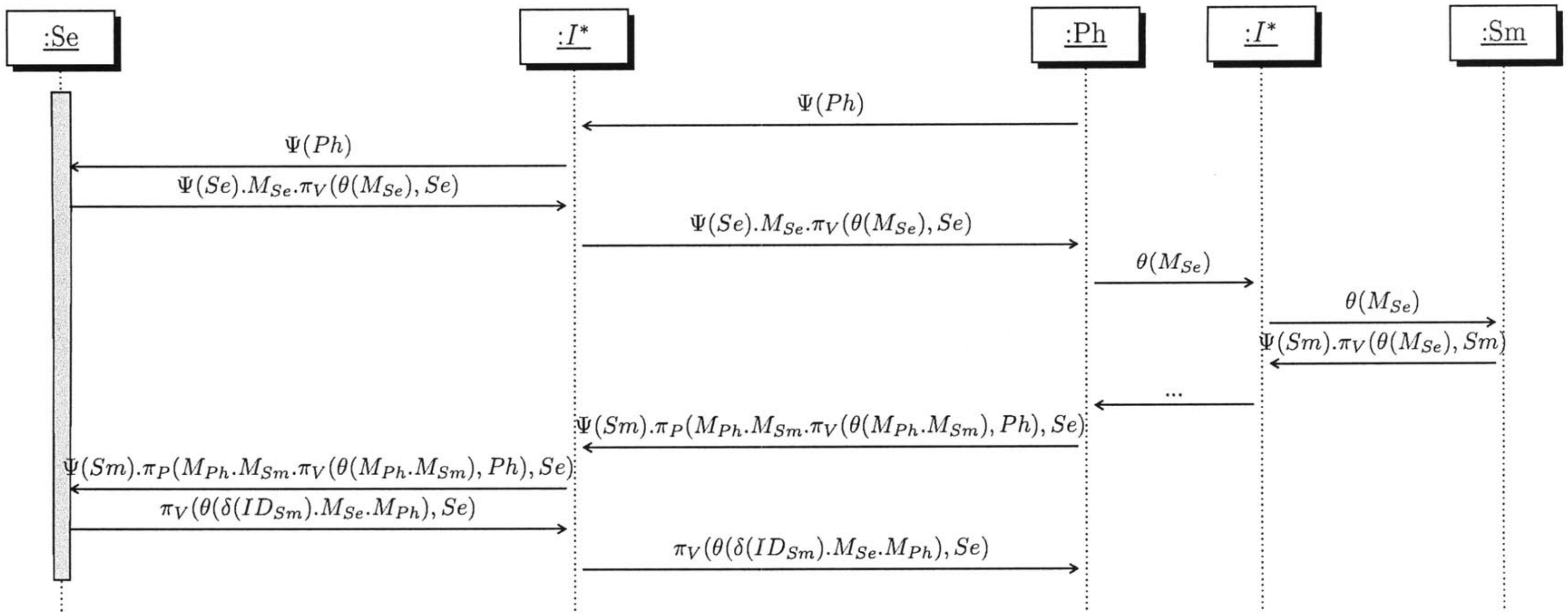

:Se
:I^*
:Ph
:I^*
:Sm
$\Psi(Ph)$
$\Psi(Ph)$
$\Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
$\Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
$\theta(M_{Se})$
$\theta(M_{Se})$
$\Psi(Sm).\pi_V(\theta(M_{Se}), Sm)$
...
$\Psi(Sm).\pi_P(M_{Ph}.M_{Sm}.\pi_V(\theta(M_{Ph}.M_{Sm}), Ph), Se)$
$\Psi(Sm).\pi_P(M_{Ph}.M_{Sm}.\pi_V(\theta(M_{Ph}.M_{Sm}), Ph), Se)$
$\pi_V(\theta(\delta(ID_{Sm}).M_{Se}.M_{Ph}), Se)$
$\pi_V(\theta(\delta(ID_{Sm}).M_{Se}.M_{Ph}), Se)$

Figure 8-2: UML sequence diagram for $\mathfrak{D}^*_{\mathfrak{B}}$ protocol, a case with a silent eavesdropper.

8.6.1 Protocol $\mathfrak{O}^*_{\mathfrak{B}}$ at a Glance

The set of initial assumptions are similar to that of $\mathfrak{O}_{\mathfrak{B}}$, except that I^* now knows the public key of certificate authority $\aleph$, Se, Ph and Sm. We thus have: $I^* \models \wp(\aleph, K_{\aleph})$, $I^* \models \wp(Se, K_{Se})$, $I^* \models \wp(Ph, K_{Ph})$ and $I^* \models \wp(Sm, K_{Sm})$. Additionally, we assume $\aleph$ and Se are trustworthy to the intruder: $I^* \models \aleph \mapsto \star$ and $I^* \models Se \mapsto \star$.

The post-conditions are the same as that of the original protocol $\mathfrak{O}_{\mathfrak{B}}$, since I^* does not alter any messages and the three principals Se, Ph and Sm participate in $\mathfrak{O}^*_{\mathfrak{B}}$ until the completion of their respective roles. The pre-conditions and the sequence of message transmissions are presented in Figures 8-3.

The sequence of transmissions now include those received and relayed by the intruder I^*. The intruder assertions we require for $\mathfrak{O}^*_{\mathfrak{B}}$ are simply that the eavesdropper does not acquire the payment details sent from the Ph:

$$\neg I^* \models \sqcap\,(I^*, M_{Ph}) \tag{8.68}$$

We assume that Sm and Se do not have any privacy requirements. Se generates a fresh M_{Se} that can be publicly visible and Sm can reveal its identity as well as $\theta(M_{Se})$.

8.6.2 PTGPA-Beta Analysis of $\mathfrak{O}^*_{\mathfrak{B}}$

We ran the PTGPA-Beta algorithm against $\mathfrak{O}^*_{\mathfrak{B}}$. The algorithm found the protocol ran to the end successfully and all post-conditions and intruder assertions were valid. The PTGPA-Beta algorithm returns true and thus $\mathfrak{O}_{\mathfrak{B}}$ is correct under attack $\mathfrak{O}^*_{\mathfrak{B}}$ by Theorem 10.

8.7 Relay Attack

In this section, we model a relay attack on $\mathfrak{O}_{\mathfrak{B}}$, denoted as $\mathfrak{O}^{\#}_{\mathfrak{B}}$. The UML sequence diagram is presented in Figure 8-4.

Seq.	Pre-Conditions	Transmission
1	$\emptyset$	$Ph \rightarrow I^* : \Psi(Ph)$
2	$\emptyset$	$I^* \rightarrow Se : \Psi(Ph)$
3	$Se \models \wp(Ph, K_{Ph})$	$Se \rightarrow I^* : \Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
4	$\emptyset$	$I^* \rightarrow Ph : \Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
5	$Ph \models \wp(Se, K_{Se})$, $Ph \models \mathfrak{S}(M_{Se}, Se)$, $Ph \models \sharp(M_{Se})$, $Ph \models \eta(M_{Se}, Se)$, $Ph \models \varpi(M_{Se}, \delta(ID_{Ph}))$	$Ph \rightarrow I^* : \theta(M_{Se})$
6	$\emptyset$	$I^* \rightarrow Sm : \theta(M_{Se})$
7	$\emptyset$	$Sm \rightarrow I^* : \Psi(Sm).\pi_V(\theta(M_{Se}), Sm)$
8	$\emptyset$	$I^* \rightarrow Ph : \Psi(Sm).\pi_V(\theta(M_{Se}), Sm)$
9	$Ph \models \wp(Sm, K_{Sm})$, $Ph \models \mathfrak{S}(M_{Se}, Sm)$, $Ph \models \eta(M_{Se}, Sm)$	$Ph \rightarrow I^* :$ $\Psi(Sm).\pi_P(M_{Ph}.M_{Sm}.\pi_V(\theta(M_{Ph}.M_{Sm}), Ph), Se)$
10	$\emptyset$	$I^* \rightarrow Se :$ $\Psi(Sm).\pi_P(M_{Ph}.M_{Sm}.\pi_V(\theta(M_{Ph}.M_{Sm}), Ph), Se)$
11	$Se \models \wp(Sm, K_{Sm})$, $Se \models \mathfrak{S}(M_{Se}, Sm)$, $Se \models \eta(M_{Se}, Sm)$, $Se \models \mathfrak{S}(M_{Ph}.M_{Sm}, Ph)$, $Se \models \eta(M_{Ph}.M_{Sm}, Ph)$	$Se \rightarrow I^* : \pi_V(\theta(\delta(ID_{Sm}).M_{Se}.M_{Ph}), Se)$
12	$\emptyset$	$I^* \rightarrow Ph : \pi_V(\theta(\delta(ID_{Sm}).M_{Se}.M_{Ph}), Se)$

Figure 8-3: Pre-conditions and message exchanges for attack protocol $\mathfrak{O}^*_{\mathfrak{B}}$.

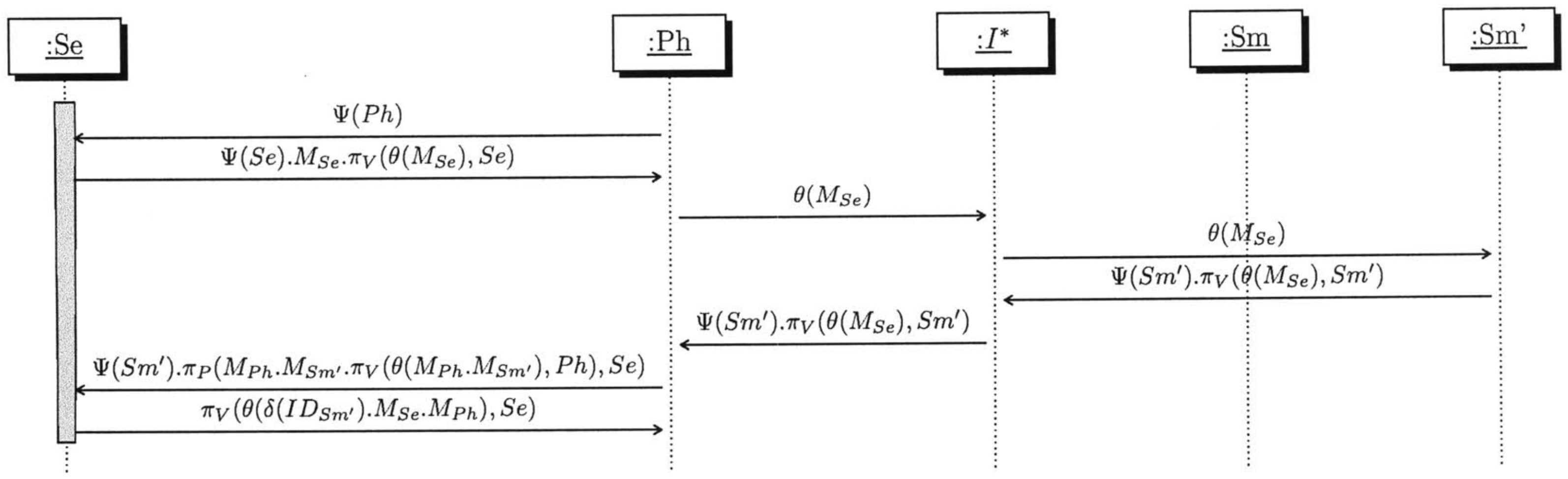

:Se
:Ph
$:I^*$
:Sm
:Sm'
$\Psi(Ph)$
$\Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
$\theta(M_{Se})$
$\theta(M_{Se})$
$\Psi(Sm').\pi_V(\theta(M_{Se}), Sm')$
$\Psi(Sm').\pi_V(\theta(M_{Se}), Sm')$
$\Psi(Sm').\pi_P(M_{Ph}.M_{Sm'}.\pi_V(\theta(M_{Ph}.M_{Sm'}), Ph), Se)$
$\pi_V(\theta(\delta(ID_{Sm'}).M_{Se}.M_{Ph}), Se)$

Figure 8-4: UML sequence diagram for attack $\mathfrak{D}^{\#}_{\mathfrak{B}}$, a case with second, authenticated smart card.

The intruder I^* prevents Sm from receiving $\theta(M_{Se})$, and I^* relays $\theta(M_{Se})$ to a second, authenticated smart card Sm' for signature. I^* then returns this signature to Ph. Ph thinks it tapped on Sm, but in fact it received tapping evidence for a different smart card, perhaps at a different geographical location. Notice that Sm is not involved in this protocol.

8.7.1 Protocol $\mathfrak{O}^{\#}_{\mathfrak{B}}$ at a Glance

The initial assumptions are the same as that of $\mathfrak{O}^{*}_{\mathfrak{B}}$. The sequence of transmissions and their pre-conditions are presented in Figure 8-5.

Seq.	Pre-Conditions	Transmission
1	$\emptyset$	$Ph \rightarrow Se : \Psi(Ph)$
2	$Se \models \wp(Ph, K_{Ph})$	$Se \rightarrow Ph : \Psi(Se).M_{Se}.\pi_V(\theta(M_{Se}), Se)$
3	$Ph \models \wp(Se, K_{Se})$, $Ph \models \mathfrak{S}(M_{Se}, Se)$, $Ph \models \sharp(M_{Se})$, $Ph \models \eta(M_{Se}, Se)$, $Ph \models \varpi(M_{Se}, \delta(ID_{Ph}))$	$Ph \rightarrow I^* : \theta(M_{Se})$
4	$\emptyset$	$I^* \rightarrow Sm' : \theta(M_{Se})$
5	$\emptyset$	$Sm' \rightarrow I^* : \Psi(Sm').\pi_V(\theta(M_{Se}), Sm')$
6	$\emptyset$	$I^* \rightarrow Ph : \Psi(Sm').\pi_V(\theta(M_{Se}), Sm')$
7	$Ph \models \wp(Sm', K_{Sm'})$, $Ph \models \mathfrak{S}(M_{Se}, Sm')$, $Ph \models \eta(M_{Se}, Sm')$	$Ph \rightarrow Se :$ $\Psi(Sm').\pi_P(M_{Ph}.M_{Sm'}.\pi_V(\theta(M_{Ph}.M_{Sm'}), Ph), Se)$
8	$Se \models \wp(Sm', K_{Sm'})$, $Se \models \mathfrak{S}(M_{Se}, Sm')$, $Se \models \eta(M_{Se}, Sm')$, $Se \models \mathfrak{S}(M_{Ph}.M_{Sm'}, Ph)$, $Se \models \eta(M_{Ph}.M_{Sm'}, Ph)$	$Se \rightarrow Ph : \pi_V(\theta(\delta(ID_{Sm'}).M_{Se}.M_{Ph}), Se)$

Figure 8-5: Pre-conditions and message exchanges for attack protocol $\mathfrak{O}^{\#}_{\mathfrak{B}}$.

Pre-conditions for message exchange 7 and 8 now relate to Sm'. This is because they were semantically linked to the smart card in message transmission 6, which is returned by an intruder that performed message redirection. By Definition 37, the set of post-conditions and intruder assertions are modified accordingly with respect to this change from the original protocol.

The set of post-conditions Σ we impose on this attack protocol consists of the following two components, $\Sigma_{(+)}$ and $\Sigma_{(-)}$:

$\Sigma_{(+)}$: Ph's assertions on the last message that Ph received. Ph asserts that it comes from Se, it is intact, and infers $\delta(ID_{Sm'}).M_{Se}.M_{Ph}$.

$$\Sigma_{(+)} = \{Ph \mid\equiv \mathfrak{S}(\delta(ID_{Sm'}).M_{Se}.M_{Ph}, Se), \tag{8.69}$$

$$Ph \mid\equiv \eta(\delta(ID_{Sm'}).M_{Se}.M_{Ph}, Se)\} \tag{8.70}$$

$\Sigma_{(-)}$: Assertions on the overall protocol at completion.

(a) Irrefutable payment from Ph:

$$Se \mid\equiv \beth(Se, \mathfrak{S}(M_{Se}, Sm')) \tag{8.71}$$

$$Se \mid\equiv \beth(Se, \eta(M_{Se}, Sm')) \tag{8.72}$$

$$Se \mid\equiv \beth(Se, \mathfrak{S}(M_{Ph}.M_{Sm'}, Ph)) \tag{8.73}$$

$$Se \mid\equiv \beth(Se, \eta(M_{Ph}.M_{Sm'}, Ph)) \tag{8.74}$$

(b) Irrefutable payment confirmation:

$$Ph \mid\equiv \beth(Ph, \mathfrak{S}(\delta(ID_{Sm'}).M_{Se}.M_{Ph}, Se)) \tag{8.75}$$

$$Ph \mid\equiv \beth(Ph, \eta(\delta(ID_{Sm'}).M_{Se}.M_{Ph}, Se)) \tag{8.76}$$

The intruder assertion Υ we impose on $\mathfrak{O}^{\#}_{\mathfrak{B}}$ is that I^* cannot reconstruct M_{Ph}:

$$\neg I^* \mid\equiv \sqcap (I^*, M_{Ph}) \tag{8.77}$$

8.7.2 PTGPA-Beta Analysis of $\mathfrak{O}_{\mathfrak{B}}^{\#}$

We ran the PTGPA-Beta algorithm against $\mathfrak{O}_{\mathfrak{B}}^{\#}$. The algorithm ran to completion successfully. All post-conditions and intruder assertions were valid. Thus by Theorem 10, $\mathfrak{O}_{\mathfrak{B}}$ is correct under attack $\mathfrak{O}_{\mathfrak{B}}^{\#}$.

It is surprising that the original protocol is deemed correct when confronted with this relay attack. The reason of this is that prior to step 5 in the original protocol $\mathfrak{O}_{\mathfrak{B}}$, Ph does not verify if the Sm is the intended merchant/goods to purchase. No pre-condition was specified in the original protocol to reflect this security requirement. Stated otherwise, Ph does not care if the tapping evidence was from Sm or some Sm'.

In practice, the application running on the phone can present a confirmation window and ask the customer to confirm that everything Ph gets back from Sm is consistent before Ph accepts the transaction. We can model this additional precaution in $TGPay$ as a pre-condition for step 5 in $\mathfrak{O}_{\mathfrak{B}}$: $Ph \models \varpi(\Psi(Sm), \delta(ID_{Sm}))$, where $\Psi(Sm)$ is the certificate of a specific Sm that Ph receives and $\delta(ID_{Sm})$ is an identity of a smart card (thus its implied goods for purchase) that is the customer's expectation.

This raises an interesting issue for some contactless payment applications such as transit ticketing, in which the server must be able to deduce the geographical location of any tap for the purposes of fare calculation and auditing. In traditional contact payment protocols, a valid transaction between a customer and a merchant can be assumed to physically take place at the merchant's reader (We assume card reader is trustworthy). That is, the bank card was physically present at the merchant's reader because of the underlying technology.

On the other hand, the contactless mobile payment protocol works in close proximity but physical contact is not required. This seperation, as demonstrated from attack $\mathfrak{O}_{\mathfrak{B}}^{\#}$, can be magnified. Therefore, in mobile contactless protocols, we can no longer deduce the physical presence of a customer from a transaction. The transaction itself can still be valid, since in the simplest configuration, only a legitimate relationship between an identified customer and merchant is demanded.

Another interesting case is when the merchant and the customer are both intruders and collude (and thus Se is interacting with two intruders). For example, Ph is dishonest and it disregards any pre-conditions before communicating with Se, after it has received a message from Sm' located at a different geographical location. Se simply verifies the non-repudiatable binding between Ph's identity and the identity of Sm', and records that Ph wanted to purchase the service as defined by $\delta(ID_{Sm'})$. Se does not require and cannot enforce, based on the limited information, that Ph was physically at Sm'.

In mobile-tag transit ticketing, customers sign-in by tapping on the smart cards mounted on platforms at stations. The payment server Se has a business requirement to verify the consistency of customer's physical location and the reported tapping evidence with any specific Sm. In this situation, the mobile phone may also transmit its authenticated GPS coordinates for server audit. We can model this consistency check performed at the payment server by introducing additional semantics in $TGPay$ such as a predicate of *closeBy(GPS_Coord, Sm)*.

In summary, we want to point out the complexity of designing these payment protocols. Protocols that are secure in one use case may not satisfy all the business and security requirements for another. However, from the protocol analysis's point of view, the TGPay proof system is flexible enough to formalize different sets of requirements, and our PTGPA framework is capable of performing verifications in full automation.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 9

Conclusion

Contents

9.1 Thesis Summary

A formal framework is developed in this thesis for automated analysis of payment protocols, based on a technique called theory generation. This technique produces a finite representation of a possibly infinite set of valid formulas that can be derived from a set of initial assumptions and a set of rules of inference. Theory generation requires a definition of preorder and some syntactical constraints to partition all rules of inference into S-Rules, G-Rules and W-Rules. With these conditions met for a proof system, validity of any given candidate formula is always decidable in finite time.

This thesis decomposes its automated analysis strategy into two parts. *TGPay* is specifically designed to model payment protocols. It provides formal semantic definitions of functions and predicates, as well as rules of inference that target secrecy, authentication, freshness, integrity, acknowledgement and non-repudiation.

Being able to validate formulas with *TGPay* using theory generation, we develop a formal approach to protocol verification called PTGPA. Our approach allows specification of pre-conditions and post-conditions besides the traditional descriptions of message transmissions in a protocol. PTGPA operates on two scenarios - Alpha-S and Beta-S. Alpha-S is one in which a candidate protocol runs without attacks from any intruders. The candidate protocol is correct iff all pre-conditions and post-conditions are met in this scenario. We model intruders' actions and knowledge sets in a second, modified protocol that represents an attack scenario. This second protocol, called Beta-S, is obtained mechanically from the original protocol, by applying a set of elementary capabilities from a Dolev-Yao intruder abstraction. We define what constitutes correctness for the original protocol under Beta-S and we show how PTGPA can be applied to verify a given protocol's correctness under both Alpha and Beta scenarios in full automation.

The thesis demonstrates the feasibility and efficacy of the proposed analysis framework in a number of case studies. We apply PTGPA to real-world payment protocols, including two contactless bank card payment protocols and the EMV SDA and DDA authentication protocols. We found security flaws of integrity, secrecy and authenticity in the two contactless bank card payment protocols and a replay attack for the EMV SDA authentication protocol.

We propose two general purpose contactless mobile payment protocols with additional desired features such as transaction non-repudiation. We verify their correctness under Alpha scenarios, with respect to a set of pre- and post-conditions that we impose. We then analyze attacks on secrecy, message relay and man-in-the-middle. We show how the two protocols successfully defend against these attacks in the context of the PTGPA Beta-S analysis framework. We discuss relevant modeling issues and their implications.

The proposed PTGPA algorithms are prototyped in Java. The implementation focuses primarily on functionality and is not optimized for performance. Figure 9-1 provides a summary of all the case studies covered in this thesis, including the size of the initial assumptions, pre- and post-conditions, intruder assertions, theory representations and verification time. The benchmarks were obtained on a x86-PC with an Intel Core i7-2640M processor (4M cache, 2.80 GHz) and 4GB of RAM.

The first column is the protocol name. The second column gives, respectively, the number of initial assumptions, total number of pre-conditions for all messages, total number of post-conditions and intruder assertions if any. The third column gives the number of formulas in the theory representation. When a protocol halts prematurely, this is the size of the theory representation at abortion. The fourth column is the time in seconds for end-to-end verification[1]. The fifth column is the correctness conclusion.

Protocol	Γ^0, Ω, Σ, Υ	**Th. Rep.**	**Time(s)**	**Correct?**
Card Type A	2-5-0-Nil	28	0.33	Alpha-S Incorrect
Card Type B	4-5-0-Nil	44	0.45	Alpha-S Incorrect
Card Type B (Secrecy Attack)	4-3-0-1	76	0.95	Beta-S Incorrect
EMV SDA	4-0-5-Nil	22	0.17	Alpha-S Incorrect
EMV DDA	7-3-0-Nil	70	0.76	Alpha-S Correct
Mobile-Reader $\mathfrak{O}_{\mathfrak{A}}$	17-12-9-Nil	164	8.4	Alpha-S Correct
Mobile-Reader $\mathfrak{O}^{*}_{\mathfrak{A}}$ (Secrecy Attack)	20-12-9-4	189	9.4	Beta-S Correct
Mobile-Reader $\mathfrak{O}^{\#}_{\mathfrak{A}}$ (Impersonation)	17-12-9-2	116	1.2	Beta-S Correct
Mobile-Tag $\mathfrak{O}^{*}_{\mathfrak{B}}$	29-15-8-Nil	205	7.4	Alpha-S Correct
Mobile-Tag $\mathfrak{O}^{*}_{\mathfrak{B}}$ (Secrecy Attack)	35-15-8-1	279	8.1	Beta-S Correct
Mobile-Tag $\mathfrak{O}^{\#}_{\mathfrak{B}}$ (Relay Attack)	41-15-8-1	231	7.8	Beta-S Correct

Figure 9-1: Protocol analyses performed in this thesis.

9.2 Thesis Contribution

This thesis makes two concrete contributions to the state-of-the-art of formal verifications of payment protocols.

[1]From the time the program takes in a protocol specification and its pre- and post-conditions, until a decision is computed.

1. Theoretical Contribution - An automated analysis framework PTGPA:

   (a) Formal analysis framework in the context of Alpha-S for correctness of original design and Beta-S for correctness when confronted with attacks.

   (b) A novel integration of theory generation with rigorous protocol analysis technique of pre- and post-conditions. This integration allows formal establishment of protocol correctness in finite time and in full automation.

   (c) *TGPay* : A comprehensive set of rules of inference is developed, including functions, predicates and rules that target payment protocols.

2. Applied Contribution - Applications of the proposed framework to real-world and newly proposed payment protocols:

   (a) Applications of PTGPA to two contactless bank card payment protocols and the EMV SDA and DDA authentication protocols. Security flaws are identified and discussed.

   (b) Proposal of two new general-purpose contactless mobile payment protocols. Their correctness under Alpha-S are verified. Attacks are examined using the Beta-S PTGPA algorithms. The proposed protocols are found secure against these attacks.

9.3 Future Research

Although the proposed analysis framework has been successfully applied to the verifications of a number of protocols, further improvements can be made in a number of areas:

1. Automatic generation of attacks: The thesis defines how an attack can be constructed from the original protocol and its pre- and post-conditions. It then checks for correctness of the original protocol under this specific attack. It is more desirable if one can automatically search for attacks from sources including, but not

limited to, a finite model of the intruder's knowledge set and a finite model of the intruder's capability. The number of distinct attack specifications for an original protocol could be potentially unbounded. It may be possible to categorize the attacks in the same manner as theory representation. One can then examine only a finite set of attacks from the representation rather than checking against each and every attack. The strategy of first constructing a sufficiently expressive set of attacks and then verifying them with a given candidate protocol is not addressed in great depth in the current literature. Future research can devote some attention in this area.

2. Performance optimization: This thesis does not consider any performance optimization of the PTGPA framework. Further studies can exploit more efficient procedures for theory generation and its applications in PTGPA. The Java implementation developed in this thesis handles commutative rewrites by exhaustively considering all possible combinations. It could become much slower when dealing with a very long sequence of message concatenations. Efficient techniques to handle rewrites such as Associative-Commutative Unification [Sti81] [Fag87] can be used. When performing membership tests, our implementation always iterates through every element in a set. Efficient data structures can be used to increase performance.

3. Enrichment of *TGPay*: The proof system *TGPay* that we present in this thesis contains rules of inference that target many important aspects of payment protocols. Further enrichments and modifications of the semantics can be made to model additional business and security requirements. The PTGPA analysis framework is flexible and is loosely decoupled with the underlying proof system.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[AC06] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1-2):2 – 32, 2006.

[AF99] C. Adams and S. Farrell. Internet X.509 public key infrastructure certificate management protocols [online], 1999. Avaiable at: $http://tools.ietf.org/html/rfc2510$.

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. *SIGPLAN Not.*, 36:104–115, Jan. 2001.

[AG97] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the Spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[AG98a] M. Abadi and A. Gordon. A bisimulation method for cryptographic protocols. *Nordic J. of Computing*, 5:267–303, Dec. 1998.

[AG98b] M. Abadi and A. Gordon. A bisimulation method for cryptographic protocols. *In ESOP'98: 7th European Symposium on Programming*, 1381:12–26, 1998.

[All09] Smart Card Alliance. Fraud in the U.S. payments industry: Fraud mitigation and prevention measures in use and chip card technology impact on fraud [online], 2009. Avaiable at: $http://www.smartcardalliance.org/resources/lib/Fraud\_EMV\_Contactless\_20091007.pdf$.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication, 1989. SRC Technical Report 39. Digital Systems Research Centre.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, February 1990.

[Cer01] I. Cervesato. The Dolev-Yao intruder is the most powerful attacker. In *Proceedings of the Sixteenth Annual Symposium on Logic in Computer Science — LICS'01*, pages 16–19. IEEE Computer Society Press. Short, 2001.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, 9:443–487, Oct. 2000.

[DJZF09] Y. Du, C. Jiang, M. Zhou, and Y. Fu. Modeling and monitoring of E-Commerce workflows. *Information Sciences*, 179(7):995 – 1006, 2009.

[DLC08] R. Dojen, I. Lasc, and T. Coffey. Establishing and fixing a freshness flaw in a key-distribution and authentication protocol. In *Intelligent Computer Communication and Processing, 2008. ICCP 2008. 4th International Conference on*, pages 185 –192, Aug. 2008.

[DSV00] L. Durante, R. Sisto, and A. Valenzano. A state-exploration technique for Spi-calculus testing equivalence verification. In *Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*, FORTE/PSTV 2000, pages 155–170, Deventer, The Netherlands, 2000. Kluwer, B.V.

[DSV03] L. Durante, R. Sisto, and A. Valenzano. Automatic testing equivalence verification of Spi calculus specifications. *ACM Trans. Softw. Eng. Methodol.*, 12:222–284, Apr. 2003.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198 – 208, Mar. 1983.

[EG83] S. Even and O. Goldreich. On the security of multi-party Ping-Pong protocols. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 34 –39, Nov. 1983.

[EMM05] S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer: grammar generation. In *FMSE*, pages 1–12, 2005.

[EMM06] S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theor. Comput. Sci.*, 367(1-2):162–202, 2006.

[EMV11a] EMVCo. The EMV 4.3 Specifications – Book 2 - Security and Key Management [online], 2011. Avaiable at: $http : //www.emvco.com/specifications.aspx?id = 223$.

[EMV11b] EMVCo. The EMV Contactless Specifications – Book D: Contactless Communication Protocol [online], 2011. Avaiable at: $http : //www.emvco.com/specifications.aspx?id = 21$.

[Exp05] Expresspay. American Express Expresspay [online], 2005. $https : //www295.americanexpress.com/cards/loyalty.do?page = expresspay$.

[Fag87] F. Fages. Associative-commutative unification. *J. Symb. Comput.*, 3(3):257–275, June 1987.

[Fin03] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification.* John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2003.

[FL09] C. Fan and Y. H. Lin. Provably secure remote truly three-factor authentication scheme with privacy protection on biometrics. *Information Forensics and Security, IEEE Transactions on*, 4(4):933 –945, Dec. 2009.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[Goo11] Google. Google Wallet [online], 2011. *http : //www.google.com/wallet.*

[GS91] K. Gaarder and E. Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3:81–98, 1991. 10.1007/BF00196790.

[HBBFJ07] T. S. Heydt-Benjamin, D. V. Bailey, K. Fu, and A. Juels. Vulnerabilities in first-generation RFID-enabled credit cards. In *in Proceedings of Eleventh International Conference on Financial Cryptography and Data Security.* Manuscript, 2007.

[HSW00] N. Hopper, S. Seshia, and J. Wing. Combining theory generation and model checking for security protocol analysis. In *CMU Department of Computer Science Report CMU-CS-00-107*, 2000.

[ID96] C. N. Ip and D. L. Dill. Verifying systems with replicated components in Murphi. In *Proceedings of the 8th International Conference on Computer Aided Verification*, CAV '96, pages 147–158, London, UK, UK, 1996. Springer-Verlag.

[IETF12a] IETF Internet Engineering Task Force. HMAC: Keyed-hashing for message authentication [online], 2012. Available at: *http : //tools.ietf.org/html/rfc*2104.

[IETF12b] IETF Internet Engineering Task Force. Internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile [online], 2012. Available at: *http : //tools.ietf.org/html/rfc*3280.

[IETF12c] IETF Internet Engineering Task Force. US Secure Hash Algorithm 1 (SHA1) [online], 2012. Available at: *http : //tools.ietf.org/html/rfc*3174.

[IETF12d] IETF Internet Engineering Task Force. Using SHA2 algorithms with cryptographic message syntax [online], 2012. Available at: $http : //tools.ietf.org/html/rfc5754$.

[IR08a] IETF-RFC5246. The Transport Layer Security (TLS) protocol version 1.2 [online], 2008. Avaiable at: $http : //tools.ietf.org/html/rfc5246$.

[IR08b] IETF-RFC6101. The Transport Layer Security (TLS) protocol version 3.0 [online], 2008. Avaiable at: $http : //tools.ietf.org/html/rfc6101$.

[ISO04] ISO/IEC-18092. Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1) [online], 2004. Avaiable at: $http : //www.iso.org/iso/catalogue_detail.htm?csnumber = 38578$.

[ISO08a] ISO/IEC-14443. Identification cards – Contactless integrated circuit cards – Proximity cards [online], 2008. Avaiable at: $http : //www.iso.org/iso/iso\_catalogue/catalogue\_ics/ catalogue\_detail\_ics.htm?csnumber = 28730$.

[ISO08b] ISO/IEC-7813. Information technology – Identification cards – Financial transaction cards [online], 2008. Avaiable at: $http : //www.iso.org/iso/iso\_catalogue/catalogue\_tc/catalogue\_detail.htm ?csnumber = 43317$.

[Kai96] R. Kailar. Accountability in electronic commerce protocols. *Software Engineering, IEEE Transactions on*, 22(5):313 –328, May 1996.

[KEAR09] K. Kostiainen, J. E. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 104–115, New York, NY, USA, 2009. ACM.

[Kin99] D. Kindred. *Theory Generation for Security Protocols.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1999.

[KP12] G. Kapoor and S. Piramuthu. Single RFID tag ownership transfer protocols. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(2):164 –173, Mar. 2012.

[KW94] V. Kessler and G. Wedel. AUTLOG - an advanced logic of authentication. In *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, pages 90 –99, Jun. 1994.

[KW97] D. Kindred and J. Wing. Closing the idealization gap with theory generation, 1997. Proceedings of the DIMACS Workshop on Cryptographic Protocol Design and Verification.

[KW99] D. Kindred and J. Wing. Theory generation for security protocols, 1999. Submitted to the ACM Transactions on Programming Languages and Systems (TOPLAS).

[LHL08] Y. Li, D. He, and X. Lu. Accountability of perfect concurrent signature. In *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*, pages 773 –777, Dec. 2008.

[LKM+05] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang. Architecture for protecting critical secrets in microprocessors. *SIGARCH Comput. Archit. News*, 33:2–13, May 2005.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, TACAs '96, pages 147–166, London, UK, UK, 1996. Springer-Verlag.

[Low98] G. Lowe. Towards a completeness result for model checking of security protocols. In *Computer Security Foundations Workshop, 1998. Proceedings. 11th IEEE*, pages 96 –105, Jun. 1998.

[Mas05] MasterCard. MasterCard PayPass [online], 2005. *http : //www.paypass.com/*.

[MCF87] J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol secuity analysis. *Software Engineering, IEEE Transactions on*, SE-13(2):274 – 288, Feb. 1987.

[Mea92] C. Meadows. Applying formal methods to the analysis of a key management protocol. In *Journal of Computer Security*, volume 1, 1992.

[Mea99] C. Meadows. Analysis of the internet key exchange protocol using the NRL protocol analyzer. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 216 –231, 1999.

[Mea03] C. Meadows. Formal methods for cryptographic protocol analysis: emerging issues and trends. *Selected Areas in Communications, IEEE Journal on*, 21(1):44 – 54, Jan. 2003.

[Mil91] R. Milner. The polyadic Pi-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[Mil99] R. Milner. *Communicating and Mobile Systems: The π-calculus.* 1999. Cambridge, UK: Cambridge University Press. ISBN 0-521-65869-1.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 141 –151, May 1997.

[MO06] J. Ma and M. A. Orgun. Trust management and trust theory revision. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(3):451 –460, May 2006.

[Mon99] D. Monniaux. Decision procedures for the analysis of cryptographic protocols by logics of belief. In *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*, pages 44 –54, 1999.

[MSnN95] A. M. Mathuria, R. Safavi-naini, and P. R. Nickolas. On the automation of GNY logic. In *Proceedings of the 18th Australian Computer Science Conference*, pages 370–379, 1995.

[NF02] NIST-FIPS197. Announcing approval of Federal Information Processing Standard (FIPS) 197, Advanced Encryption Standard (AES) [online], 2002. Avaiable at: $http : //csrc.nist.gov/archive/aes/frn - fips197.pdf$.

[NF12a] NFC-Forum. NFC digital protocol technical specification [online], 2012. Avaiable at: $http : //www.nfc - forum.org/specs/spec\_list/$.

[NF12b] NFC-Forum. NFC Logical Link Control Protocol (LLCP) technical specification [online]. 2012. Avaiable at: $http : //www.nfc - forum.org/specs/spec\_list/$.

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *In 8th IEEE Computer Security Foundations Workshop*, pages 98–107. Press, 1995.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. *Theoretical Computer Science*, pages 174–190, 2001.

[RT03] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theor. Comput. Sci.*, 299(1-3):451–475, April 2003.

[San96] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996.

[SD97] U. Stern and D. L. Dill. Parallelizing the Murphi verifier. In *Computer Aided Verification. 9Th International Conference*, pages 256–267. Springer-Verlag, 1997.

[SLBS08] A. C. Squicciarini, W. Lee, E. Bertino, and C. X. Song. A policy-based accountability tool for grid computing systems. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pages 95 –100, Dec. 2008.

[SM09] S. Y. Shi and Y. M. Mao. An improvement key distribution protocol and its BAN analysis. In *Future Computer and Communication, 2009. ICFCC 2009. International Conference on*, pages 381 –384, Apr. 2009.

[SSLH11] S. Sundareswaran, A. Squicciarini, D. Lin, and S. Huang. Promoting distributed accountability in the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 113 –120, Jul. 2011.

[Sti81] M. E. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28(3):423–434, July 1981.

[SvO94] P. F. Syverson and P. C. van Oorschot. On unifying some cryptographic protocol logics. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 14 –28, May 1994.

[SY08] Sufatrio and R. Yap. Extending BAN logic for reasoning with modern PKI-based protocols. In *Network and Parallel Computing, 2008. NPC 2008. IFIP International Conference on*, pages 190 –197, Oct. 2008.

[Vis07] Visa. Visa payWave [online], 2007. $http : //usa.visa.com/personal/cards/paywave/$.

[vO93] P. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 232–243, New York, NY, USA, 1993. ACM.

[ZF03] H. Zhou and S. N. Foley. Fast automatic synthesis of security protocols using backward search. In *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, FMSE '03, pages 1–10, New York, NY, USA, 2003. ACM.

Appendix A

Proofs for Chapter 3

Proof of Lemma 1. Let X denote the finite set of variables from ϕ_1 that are to be replaced by σ. Because all variables from X are implicitly universally quantified by definition, it follows that we can use only instantiation to show that:

$$\phi_1 \vdash_W \sigma\phi_1 \tag{A.1}$$

Prepending this before $\sigma\phi_1 \vdash_W \sigma\phi_2$, we have the desired result. □

Proof of Lemma 2. By *unify* Definition 8, it follows that $\sigma\phi_1 = \sigma\phi_2$. By Lemma 1, we have the proof. □

Proof of Lemma 3. Because ϕ_1 is grounded, all variables that are to be replaced in σ are from ϕ_2. By *unify* Definition 8, it follows that $\phi_1 = \sigma'\phi_2$ modulo rewrites. Let $\mathbb{X}$ be the set of free variables appeared in ϕ_2. Clearly, for any variable $X \in \mathbb{X}$, it is replaced with the same grounded term in both σ and σ'. By definition 7, σ is an extension of σ' with respect to ϕ_2.

□

Proof of Theorem 1 [Kin99]. We proceed with an induction on the recursion depth of the function *backward_chain*() in Algorithm 4. First we consider the base case. When function *backward_chain*() is immediately returned and thus its recursion depth is zero, it must be the case that $goal = \emptyset$. In this case $\emptyset$ is returned and the theorem is trivially satisfied.

Supposing the theorem holds for any recursion depth of the *backward_chain*() function that is less than N, we want to show that it still holds when exactly N calls are made. Now suppose the *backward_chain*() function has chosen the first goal g (denote the rest of the goals as *other_goals* such that $\Phi = \{g, other\_goals\}$), and has invoked *backward_chain_one*(g, Γ). There are two cases:

- There exists a formula $\phi \in \Gamma$ such that $\sigma_1 = unify(g, \phi)$. By Lemma 2, we have $\phi \vdash_W \sigma_1 g$ and thus $\Gamma \vdash_W \sigma_1 g$. By induction, we have $\Gamma \vdash_{GW} \sigma_2 \circ \sigma_1 other\_goals$. Hence for this case we have the desired result.

- Otherwise, *reverse_apply_grule*(G, g, Γ) is invoked. It follows that σ_3 is a unifier of g and G's conclusion. By definition of unification, we have

$$\sigma_3 Conclusion(G) \vdash_W \sigma_3 g \tag{A.2}$$

  backward_chain is then invoked (with premises of G instantiated with σ_3) to return σ_4. This invocation contains less than N calls of *backward_chain*, and by induction, we have:

$$\Gamma \vdash_{GW} \sigma_4(\sigma_3 Premises(G)) \tag{A.3}$$

  Combining A.2 and A.3, we have now,

$$\Gamma \vdash_{GW} \sigma_4 \sigma_3 g \tag{A.4}$$

$$\Gamma \vdash_{GW} \sigma_1 g \tag{A.5}$$

Similar to the first case, we have:

$$\Gamma \vdash_{GW} \sigma_2 \circ \sigma_1 other\_goals \tag{A.6}$$

$$\Gamma \vdash_{GW} \sigma\Phi \tag{A.7}$$

□

Proof of Theorem 2 [Kin99]. Similar to the previous proof, we proceed with an induction on the recursion depth of the *closure*() function in Algorithm 2. The base case is when the recursion depth is zero and $fringe = \emptyset$. Since the proof required no application of any S-Rules, i.e. $\Gamma \vdash_W \Gamma$, the base case is valid.

Next we assume the theorem holds when there are less than N recursive calls to the *closure*() function, and we want to show that the theorem still holds when exactly N recursive calls to the *closure*() function need to be invoked.

When calling the *closure*() function the first time, $fringe'$ collects new formulas ϕ, each of which is of the form of $\sigma(Conclusion(S))$, for some instantiation σ and an S-Rule S. By Theorem 1, there exists a proof $\mathfrak{P}'$ for all $\sigma\phi$ where $\phi \in Premises(S)$, using instantiation, G-Rules and W-Rules only. We can add the application of this last S-Rule to $\mathfrak{P}'$ and thereby write a full proof $\mathfrak{P}$ of $\sigma(Conclusion(S))$. We have thus shown that for any new formula ϕ derived, its last rule of application is an S-Rule and it holds that $(\Gamma \cup fringe) \vdash \phi$.

The closure function then recursively calls $closure(\Gamma', fringe')$, and eventually returns the grand theory representation $\Gamma^{\#}$ (We show the termination proof later). By induction, for each $\phi^* \in \Gamma^{\#}$ newly derived in subsequent invocations, there is a valid proof of ϕ^*, whose last rule of application is an S-Rule. □

Proof of Theorem 3 [Kin99]. We proceed with an induction on the total number of G-Rules applied to prove Φ in $\mathfrak{P}^*$. Suppose when G-Rules are applied less than N times, the theorem holds; we prove that it still holds when exactly N G-Rule applications are used in $\mathfrak{P}^*$.

Without loss of generality, we consider an arbitrary ϕ_i and its corresponding proof $\mathfrak{P}_i$ such that:

$$\Gamma \vdash^{\mathfrak{P}_i} \sigma'\phi_i \tag{A.8}$$

Then clearly there is a corresponding action in *backward_chain*, where:

$$\sigma_1 \in \textit{backward\_chain\_one}(\phi_i, \Gamma) \tag{A.9}$$

is invoked. Looking into the *backward_chain_one*(ϕ_i, Γ) function, there are two cases: we either find a direct instantiation with any $\phi \in \Gamma$, or we need to apply a G-Rule to continue on prove the premises of that G-Rule. For this proof we can safely ignore rewrites in this function because it is size-preserving and does not incur any G-Rule application. There are two cases:

- Case A) ϕ_i can be unified with some formula $\phi \in \Gamma$. Thus, $\phi \vdash_W \sigma'\phi_i$ and $\sigma_1 = unify(\phi, \phi_i)$. By Lemma 3, we have that σ' is an extension of σ_1 with respect to ϕ_i. Since $\sigma = \sigma_2 \circ \sigma_1$ and $\sigma_1\phi_i$ is grounded, we have that σ' is an extension of σ with respect to ϕ_i. We have therefore reduced the problem to one less premise with the same number of G-Rule applications.

- Case B) $\mathfrak{P}_i$ contains at least one G-Rule application. The recursive call to *backward_chain* (which computes σ_2) takes a partially instantiated subset of Φ (all but ϕ_i). Since the proof of this subset contains less than N G-Rule applications, we can apply the induction. It is sufficient to show that if σ' is an extension of some σ_1 returned by calling *reverse_apply_grule*(G, ϕ_i, Γ), with respect to ϕ_i, the theorem holds. It remains to demonstrate this.

Function *reverse_apply_grule*(G, ϕ_i, Γ) takes in a G-Rule G. Since G is the last rule (a G-Rule) applied in $\mathfrak{P}_i$, $\mathfrak{P}_i$ can thus be partitioned in the following way:

$$\Gamma \vdash_{GW} \sigma'(Premises(G)) \tag{A.10}$$

$$\sigma'(Premises(G)) \vdash_{G} \sigma'(Conclusion(G)) \tag{A.11}$$

Because the proof $\Gamma \vdash_{GW} \sigma'(Premises(G))$ requires less than N G-Rule applications, we can invoke the induction. The remainder of the proof shows that σ' is an extension of σ_1, which by definition is $\sigma_4 \circ \sigma_3$ and is done by cases.

Consider an arbitrary term X of ϕ_i and focus all substitutions on this term. Let P and Q be two predicate symbols, the G-Rule structurally be:

$$\frac{Q(A)}{P(A,\_)} \tag{A.12}$$

Where "_" denotes the rest of arguments for predicate P. We now consider the following possible cases:

1. Term X is a variable and its unification with the conclusion of G is propagated to the premises of G, for example $\phi_i = P(X,\_)$. Without loss of generality define $\sigma' = \{X \rightarrow x\}$ for a grounded term x. Since σ_3 works on ϕ_i, we have that $\sigma_3 = \{X \rightarrow A\}$. Since G is the last rule applied it follows that we must also have proof of $\Gamma \vdash_{GW} \sigma^* Q(A)$ for some $\sigma^* = \{A \rightarrow x\}$. By induction, σ^* is an extension of σ_4, and therefore either $\sigma_4 = \{A \rightarrow x\}$ or $\sigma_4 = \{A \rightarrow Z\}$ for a variable Z. It then follows that either $\sigma_4 \circ \sigma_3 = \{X \rightarrow x\}$ or $\sigma_4 \circ \sigma_3 = \{X \rightarrow Z\}$. In either cases, we have that σ' is an extension of $\sigma_1 = \sigma_4 \circ \sigma_3$ with respect to X.

2. Term X is a variable and its unification with the conclusion of G will not be propagated to the premises of G, for example, $\phi_i = P(\_, X)$. Again let $\sigma' = \{X \rightarrow x\}$ for a grounded term x. Since G must be applied, it follows that $\sigma_3 = \sigma' = \{X \rightarrow x\}$ and σ_4 does not replace X. Putting everything together, one can conclude that σ' is an extension of $\sigma_1 = \sigma_4 \circ \sigma_3$ with respect to X.

3. Term X is a grounded term. In this case both σ' and σ_3 are "emptyset" with

respect to this term, so we have the desired result trivially.

Since term X from ϕ_i is arbitrary , we have proved that σ' is an extension of $\sigma_1 = \sigma_4 \circ \sigma_3$. This completes the entire completeness proof for *backward_chain* function.

$\square$

Proof of Theorem 4 [Kin99]. We proceed with an induction proof on the recursion depth of the *closure*() function in Algorithm 2. If the depth is 0, then Γ is returned from *closure*() function because $fringe = \emptyset$. Looking at the proof $\mathfrak{P}$ from Eq. (3.9) which generates formula ϕ, we found that this proof contains at least one S-Rule application from $(\emptyset \cup \Gamma)$. Since $\{\emptyset, \Gamma\}$ is a partial theory representation, it follows that there exists a formula ϕ^* where $\Gamma \vdash^{\mathfrak{P}^*} \phi^*$, and the last rule of application in $\mathfrak{P}^*$ is an S-Rule, and $(\emptyset \cup \Gamma) \vdash_W \phi^*$, which is impossible. This implies that some ϕ from Eq. (3.9) does not exist, the base case is thus valid.

Now suppose the theorem holds when the *closure*()'s recursion depth is less than N. We show that it still holds when the depth is precisely N. Since $closure(fringe, \Gamma)$ returns $closure(fringe', \Gamma')$, if we can prove that $\{fringe', \Gamma'\}$ is also a partial theory representation, then we are done by the inductive hypothesis. It remains to demonstrate so.

Let F be a formula such that it can be proved from Γ' using only one S-Rule S, which is the last rule of application. Let σ be the substitution under which $F = \sigma Conclusion(S)$. By Theorem 3, we have that σ is an extension of some σ^* where:

$$\sigma^* = backward\_chain(premises(S), \Gamma') \tag{A.13}$$

Thus,

$$apply\_srule(S, \Gamma') = \sigma^* Conclusion(S) \tag{A.14}$$

$$apply\_srule(S, \Gamma') \vdash_W \sigma Conclusion(S) \tag{A.15}$$

$$fringe' \cup \Gamma' \vdash_W \sigma Conclusion(S) \tag{A.16}$$

By the definition of F, we have that $fringe' \cup \Gamma' \vdash_W F$. Thus $\{fringe', \Gamma'\}$ is a partial theory representation. □

Proof of Lemma 4 [Kin99]. By the substitution property of $\preceq$, we have that for any σ, $\sigma\phi_2 \preceq \sigma\phi_1$. Also $\sigma\phi_1 \preceq \phi^* \in \Gamma$ and by transitivity we have $\sigma\phi_2 \preceq \phi^* \in \Gamma$, the desired result. □

Proof of Lemma 5 [Kin99]. We prove this by cases. If ϕ_2 is a rewrite of ϕ_1, then by the definition of rewrite, $\phi_2 \preceq \phi_1$. By the previous Lemma, this case is proved. Now suppose ϕ_2 is proved using substitution, that is $\phi_2 = \sigma\phi_1$. Then for any σ_2, it must be the case that $(\sigma_2 \circ \sigma)\phi_1$ is size-bounded by Γ, this implies that $\sigma\phi_1$ is size-bounded by Γ, so this case is also proved. □

Proof of Lemma 6 [Kin99]. Let $\phi^* \in \Gamma$ and let an arbitrary ϕ be size-bounded by ϕ^*. Then it follows that $\phi \preceq \phi^*$. By the definition of $\preceq$, there exists a finite number of such ϕ with respect to variable renaming. Because there also exist a finite number of such ϕ^*, the Lemma is proved. □

Proof of Lemma 7 [Kin99]. Let C be the conclusion of G and P be the premises of G. By the definition of G-Rule, $P_i \preceq C$ for any $P_i \in P$, and hence $\sigma_3 P_i \preceq \sigma_3 C$. Looking at the *reverse_apply_grule* function we found that $\sigma_3 = unify(\phi, C)$. By Lemma 2, this implies $\phi \vdash_W \sigma_3 C$. By Lemma 5, $\sigma_3 C$ is also size-bounded by Γ. Finally, by transitivity of the $\preceq$ operator, $\sigma_3 P_i$ is size bounded by Γ. □

Proof of Lemma 8 [Kin99]. In order for the function to return, there must exists some $\phi^{ground} \in \Gamma$ such that $\sigma' = unify(\phi, \phi^{ground})$. It follows from Lemma 2 that $\phi^{ground} \vdash_W \sigma'\phi$. Since ϕ^{ground} is size-bounded by Γ^0, by Lemma 5, so is $\sigma'\phi$. Because $\sigma = \sigma'$, we are done. □

Proof of Lemma 9. We partition Φ into primary premises Φ_p and side conditions (non-primary) Φ_s. The goal selection function will always select $\phi_p \in \Phi_p$ for resolution first before moving on to side conditions. Suppose at some stage $\phi_p \in \Phi_p$ is selected, then by Lemma 8,

$$backward\_chain\_one(\phi_p, \Gamma) \tag{A.17}$$

will halt. This is because ϕ_p does not unify with any G-Rule conclusion by S/G restriction, and unification with any $\phi^{ground} \in \Gamma$ is done in finite time.

Because Φ_s is finite, it is now sufficient to show that for each of the remaining $\phi_s \in \Phi_s$, a call to *backward_china_one* with ϕ_s as the target must halt. Without loss of generality, let σ^* be the accumulative substitution immediately before ϕ_s. If we can prove that $reverse\_apply\_grule(G, \sigma^*\phi_s, \Gamma)$ always terminates, we are done. It remains to show that this is the case.

Using Lemma 6, we observe that Γ is finite modulo variable renaming, since it is size-bounded by a finite set Γ^0. By the second condition of S/G restriction, there exists a primary condition ϕ_p such that $\phi_s \preceq \phi_p$. By the property of $\preceq$, one can deduce that $\sigma^*\phi_s \preceq \sigma^*\phi_p$. Since each $\sigma^*\phi_p$ is size-bounded by Γ, $\sigma^*\phi_s$ is also size-bounded by Γ. By Lemma 7, any formulas passed to *backward_chain*() from *reverse_apply_grule*() are also size-bounded by Γ. By Lemma 6, there are finite numbers of formulas that are size-bounded by Γ, and *backward_chain*() only returns substitutions corresponding to this finite set of formulas. Hence call to *backward_chain*() from *reverse_apply_grule*() function will terminate and hence $reverse\_apply\_grule(G, \sigma^*\phi_s, \Gamma)$ terminates. □

Proof of Lemma 10 [Kin99]. Let C be the conclusion of S and let P be a primary premise of S. Thus we have $C \preceq P$. Since P does not match any G-Rule conclusion, applying Lemma 8, we have for every σ' such that $\sigma' \in backward\_chain\_one(P, \Gamma)$, $\sigma' P$ is size-bounded by Γ^0. Now by the property of $\preceq$, $\sigma' C$ is also size-bounded by Γ^0. □

Proof of Lemma 11 [Kin99]. We show that the size-boundedness is preserved under recursive self-calling. If $fringe$ and Γ are, at the start, size-bounded by some finite set Γ^0 then Γ' must also be size-bounded, since $\Gamma' = fringe \cup \Gamma$. Moreover, formulas returned by $apply\_srule(S, \Gamma')$ are size-bounded by Γ^0 by Lemma 10 and terminates in finite time by Lemma 9. This leads to the conclusion that $fringe'$ is also size-bounded by Γ^0 with respect to renaming. So we have shown that *fringe'* and Γ' are both size-bounded by some finite set Γ^0. By Lemma 6, there are finite number of formulas bounded by Γ^0, so *closure*() must halt after exploring all distinct ones. □

Proof of Theorem 5. The initial assumptions Γ^0 passed to *closure*() function are finite and grounded. Therefore it is size-bounded by itself. By Lemma 11, *theory_gen*() will terminate. □

Proof of Theorem 6. Each time $derivable(\phi, \Gamma^{\#})$ is called, it returns if we either find $\phi \in \Gamma^{\#}$ or ϕ does not unify with conclusions of any G-Rule. In either case the function returns immediately. Otherwise, the function recursively calls itself with any formula of its first argument, denoted as ϕ', such that $\phi' \preceq \phi$. By Definition 11, there are finite number of such ϕ'. Hence the recursion depth is finitely bounded. Thus we have found that the function either terminates immediately or calls itself recursively for finite number of times. Therefore it must terminate in finite time. □

Proof of Theorem 7. First, if the prerequisites as defined in Definition 25 are satisfied, function *theory_gen*() must stop in finite time by Theorem 5. Next we show that it is both sound and complete if it terminates. Suppose ϕ is a formula returned by the function, such that $\phi \in closure(\Gamma^0, \{\})$. By Theorem 2, there exists a proof $\mathfrak{P}$ such that $\Gamma^0 \vdash^{\mathfrak{P}} \phi$ and the last rule of application in $\mathfrak{P}$ is an S-Rule. Therefore every formula returned by the function is (R, S_Rule) theory-representation sound.

Now let ϕ be a formula such that $\Gamma^0 \vdash^{\mathfrak{P}} \phi$ and the last rule of application in $\mathfrak{P}$ is an S-Rule. By Theorem 4, there exists a formula ϕ^* such that,

$$\phi^* \in closure(\Gamma^0, \emptyset) \tag{A.18}$$

and $\phi^* \vdash_W \phi$. Therefore any formula that is in the (R, S_Rule) theory representation deduced from Γ^0 can be proved using only instantiation and rewrites from some ϕ^* returned by the *closure*() function. We have that the *theory_gen*() function terminates in finite time and that it is both sound and complete. □

Proof of Corollary 1. Suppose $\Gamma^0 \vdash \phi$, then there must be a sequence of deductions using R from Γ^0 till arriving at ϕ. Let ϕ^* be any formula from the sequence such that it is derived from an S-Rule S. Then by the completeness of *closure*(), Theorem 4, $\phi^* \in \Gamma^{\#}$. Since we also have that $\Gamma^{\#} \vdash_{GW} \sigma'\phi$, it follows from Theorem 3 that for some $\sigma \in backward\_chain(\phi, \Gamma^{\#})$, σ' is an extension of σ with respect to ϕ. Since ϕ is grounded, we have that $\sigma' = \emptyset$ and therefore $backward\_chain(\phi, \Gamma^{\#})$ cannot return an emptyset.

Now we proceed with the reverse direction. If $backward\_chain(\phi, \Gamma^{\#})$ returns non-emptyset, then by Theorem 1, there exists a proof $\mathfrak{P}$ of ϕ from $\Gamma^{\#}$ using G-Rules or W-Rules only such that $\Gamma^{\#} \vdash^{\mathfrak{P}}_{GW} \phi$. By Theorem 7, $\Gamma^{\#}$ is a theory representation generated from Γ^0 using R, thus for any $\phi' \in \Gamma^{\#}$, $\Gamma^0 \vdash^{\mathfrak{P}'} \phi'$. We can thus construct a proof of ϕ from Γ^0 using R, by concatenating some $\mathfrak{P}'$ with $\mathfrak{P}$. Thus $\Gamma^0 \vdash \phi$ by Definition 18 and ϕ is valid with respect to Γ^0 and R by Definition 17. $\square$

Appendix B

Proofs for Chapter 4

Proof of Lemma 12. Since Φ is finite, it is sufficient to show that each call to the function *derivable*$(\phi, \Gamma^{\#})$ terminates in finite time, where ϕ is grounded. By Lemma 6, we have the desired result. $\square$

Proof of Theorem 8. Each call to *theory_gen*() must terminate by Theorem 5 and each call to *allValid*() must terminate by Lemma 12. Since the set of pre-conditions of any message transmission, the set of post-conditions and the intruder assertions, contain finite number of grounded formulas; By Definition 31, PTGPA-Alpha and PTGPA-Beta also consist of a finite number of calls to either *theory_gen*() or *allValid*(), and therefore they terminate in finite time. $\square$

Proof of Lemma 13. By Corollary 1, *derivable*$(\phi, \Gamma^{\#}) = \emptyset$ iff ϕ is invalid for any $\phi \in \Phi$. If allValid() returns false, then there is a $\phi \in \Phi$ that is not valid. If allValid() returns true, then for any $\phi \in \Phi$, *derivable*$(\phi, \Gamma^{\#})$ returns non-emptyset. By Corollary 1, any such ϕ is valid. $\square$

Proof of Lemma 14. We prove any formula from Γ^k is in Γ^k_* and that the reverse also holds. Let ϕ be a formula from Γ^k. By soundness part of Theorem 7, there exists a proof $\mathfrak{P}$ such that $\Phi^{k-1} \cup \phi_{k-1} \vdash^{\mathfrak{P}} \phi$ and the last rule of application in $\mathfrak{P}$ is an S-Rule. Since Γ^k_* is obtained from *theory_gen*$(\Gamma^{k-1} \cup \phi_{k-1}, R, \preceq)$, and $\Phi^{k-1} \subseteq \Gamma^{k-1}$, therefore $\phi \in \Gamma^k_*$.

Consider the reverse, that ϕ is now a formula from Γ^k_*. By soundness part of Theorem 7, there exists a proof $\mathfrak{P}$ such that:

$$\Gamma^{k-1} \cup \phi_{k-1} \vdash^{\mathfrak{P}} \phi \tag{B.1}$$

and the last rule of application in $\mathfrak{P}$ is an S-Rule. Similarly, by the definition of Γ^{k-1}, there exists a proof $\mathfrak{P}'$ such that:

$$\phi_1 \cup ... \cup \phi_{k-2} \vdash^{\mathfrak{P}'} \Gamma^{k-1} \tag{B.2}$$

Append $\mathfrak{P}'$ to $\mathfrak{P}$ we have a new proof that:

$$\phi_1 \cup ... \cup \phi_{k-2} \cup \phi_{k-1} \vdash^{\mathfrak{P}'+\mathfrak{P}} \phi \tag{B.3}$$

By the completeness part of Theorem 7, $\phi \in \Gamma^k$. □

Proof of Theorem 9. The proof follows directly from Lemmas 13 and 14. If Alpha-S PTGPA returns true for a given protocol $\mathfrak{D}$, then by Lemma 14, for any $k \leq n$, Γ^k is a theory representation generated from $\{\Gamma^0 \cup \mathfrak{D}_1 \cup ... \cup \mathfrak{D}_{k-1}\}$ using rules of inference R. By Lemma 13, every formula in Ω_k is valid with respect to $\{\Gamma^0 \cup \mathfrak{D}_1 \cup ... \cup \mathfrak{D}_{k-1}\}$ and R. By similar reasonings, every formula in Σ is valid with respect to $\{\Gamma^0 \cup \mathfrak{D}\}$ and R. Therefore $\mathfrak{D}$ is Alpha-S correct. Now suppose Alpha-S PTGPA returns false given $\mathfrak{D}$. Either for some $k \leq n$, allValid() returns false, so that there exists a formula in Ω_k that is not valid with respect to $\{\Gamma^0 \cup \mathfrak{D}_1 \cup ... \cup \mathfrak{D}_{k-1}\}$ and R; Or there exists a formula in Σ that is not valid with respect to $\{\Gamma^0 \cup \mathfrak{D}\}$ and R. In either case $\mathfrak{D}$ is invalid as defined in Definition 39, as can be shown by Lemma 13. We have thus shown that if Alpha-S PTGPA returns true, then $\mathfrak{O}$ is Alpha-S correct and if Alpha-S PTGPA returns false, then $\mathfrak{O}$ is Alpha-S incorrect. □

Proof of Theorem 10. If Beta-S PTGPA returns true for a given attack $\mathfrak{O}^*$, its initial assumption Γ^0, pre-conditions Ω^*, post-conditions Σ^* and intruder assertions Υ, then there are three possible cases to consider:

1. For some k such that $1 \leq k \leq n$, $\Gamma^k = theory\_gen(\Gamma^{k-1} \cup \mathfrak{O}^*_{k-1}, R, \preceq)$, and $allValid(\Omega^*_k, \Gamma^k, R)$ returns false and $allValid(\Upsilon, \Gamma^k, R)$ returns true. By lemma 14,

$$\Gamma^k = theory\_gen(\Gamma^0 \cup \mathfrak{O}^*_1, ..., \cup \mathfrak{O}^*_{k-1}, R, \preceq) \tag{B.4}$$

By lemma 13, there is a $\phi^{Pre} \in \Omega^*_k$ and for every $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{O}^*_1, ..., \cup \mathfrak{O}^*_{k-1} \nvdash \phi^{Pre} \tag{B.5}$$

$$\Gamma^0 \cup \mathfrak{O}^*_1, ..., \cup \mathfrak{O}^*_{k-1} \vdash \phi^{Intruder} \tag{B.6}$$

By Definition 40, $\mathfrak{O}$ is correct under attack $\mathfrak{O}^*$ for this case.

2. $allValid(\Sigma^*_{(+)}, \Gamma^{\#}, R)$ returns false and $allValid(\Upsilon, \Gamma^{\#}, R)$ returns true. Then by lemma 13, there exists a $\phi^{Last} \in \Sigma^*_{(+)}$ such that

$$\Gamma^0 \cup \mathfrak{O}^* \nvdash \phi^{Last} \tag{B.7}$$

and for each and every formulas $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{O}^* \vdash \phi^{Intruder} \tag{B.8}$$

Hence by Definition 40, $\mathfrak{O}$ is correct under attack $\mathfrak{O}^*$ for this case.

3. Else, $allValid(\Sigma^*_{(-)}, \Gamma^{\#}, R)$ returns true and $allValid(\Upsilon, \Gamma^{\#}, R)$ returns true. By lemma 13, for each and every $\phi^{Post} \in \Sigma^*_{(-)}$,

$$\Gamma^0 \cup \mathfrak{O}^* \vdash \phi^{Post} \tag{B.9}$$

and, for each and every $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{O}^* \vdash \phi^{Intruder} \tag{B.10}$$

By Definition 40, $\mathfrak{D}$ is correct under attack $\mathfrak{D}^*$ for this case.

Now we consider the reverse. Suppose Beta-S PTGPA returns false. There are only four possible ways the algorithm returns false:

1. For some k such that $1 \leq k \leq n$, $\Gamma^k = \mathit{theory\_gen}(\Gamma^{k-1} \cup \mathfrak{D}^*_{k-1}, R, \preceq)$, and $\mathit{allValid}(\Omega^*_k, \Gamma^k, R)$ returns false and $\mathit{allValid}(\Upsilon, \Gamma^k, R)$ returns false. We have for some $\phi^{Pre} \in \Omega^*_k$ and some $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{D}^*_1 \cup ... \cup \mathfrak{D}^*_{k-1} \nvdash \phi^{Pre} \tag{B.11}$$

$$\Gamma^0 \cup \mathfrak{D}^*_1 \cup ... \cup \mathfrak{D}^*_{k-1} \nvdash \phi^{Intruder} \tag{B.12}$$

   By Definition 40, $\mathfrak{D}$ is incorrect under attack $\mathfrak{D}^*$ for this case.

2. $\mathit{allValid}(\Sigma^*_{(+)}, \Gamma^{\#}, R)$ returns false and $\mathit{allValid}(\Upsilon, \Gamma^{\#}, R)$ returns false. Then by lemma 13, there exists a $\phi^{Last} \in \Sigma^*_{(+)}$ such that

$$\Gamma^0 \cup \mathfrak{D}^* \nvdash \phi^{Last} \tag{B.13}$$

   and for some formulas $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{D}^* \nvdash \phi^{Intruder} \tag{B.14}$$

   Hence by Definition 40, $\mathfrak{D}$ is incorrect under attack $\mathfrak{D}^*$ for this case.

3. $\mathit{allValid}(\Sigma^*_{(-)}, \Gamma^{\#}, R)$ returns false or $\mathit{allValid}(\Upsilon, \Gamma^{\#}, R)$ returns false. By lemma 13, for some $\phi^{Post} \in \Sigma^*_{(-)}$,

$$\Gamma^0 \cup \mathfrak{D}^* \nvdash \phi^{Post} \tag{B.15}$$

   or, for some $\phi^{Intruder} \in \Upsilon$,

$$\Gamma^0 \cup \mathfrak{D}^* \nvdash \phi^{Intruder} \tag{B.16}$$

By Definition 40, $\mathfrak{D}$ is incorrect under attack $\mathfrak{D}^*$ for this case. We have shown that if Beta-S PTGPA returns true, then $\mathfrak{D}$ is Beta-S correct for attack $\mathfrak{D}^*$ and if Beta-S PTGPA returns false, then $\mathfrak{D}$ is Beta-S incorrect for attack $\mathfrak{D}^*$.

$\square$

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix C

Proofs for Chapter 5

Proof of Lemma 15. The only two forms of substitutions over an atomic variable, observing the syntactical constraints defined above, are: substitution of the atomic variable with a 0-ary function and renaming the atomic variable. The first form does not change $\Pi(\phi)$, since neither the atomic variable before substitution nor the constant after the substitution counts. The renaming only changes the variable name but still retains its atomic variable property.

Since σ only replaces atomic variables, therefore for any non-atomic variables X, $\Lambda(\phi, X) = \Lambda(\sigma\phi, X)$. ☐

Proof of Lemma 16. We shall show that $\preceq_{TGPay}$ is both reflexive and transitive. Let ϕ_1, ϕ_2, ϕ_3 be formulas. $\phi_1 \preceq_{TGPay} \phi_1$ clearly holds so $\preceq_{TGPay}$ is reflexive. We next show $\preceq_{TGPay}$ is transitive.

Suppose $\phi_1 \preceq_{TGPay} \phi_2$ and $\phi_2 \preceq_{TGPay} \phi_3$. Since $\Pi(\phi_1) \leq \Pi(\phi_2)$ and $\Pi(\phi_2) \leq \Pi(\phi_3)$, we have that $\Pi(\phi_1) \leq \Pi(\phi_3)$, so the condition on Alphabetic Count is met.

As for Variable Frequency, ϕ_1 only contains non-atomic variables that appear in ϕ_2 and ϕ_2 only contains non-atomic variables that appear in ϕ_3. It then follows that for any non-atomic variable X from ϕ_1, $\Lambda(\phi_1, X) \leq \Lambda(\phi_2, X)$ and also that $\Lambda(\phi_2, X) \leq \Lambda(\phi_3, X)$. Hence, for any X appearing in ϕ_1, $\Lambda(\phi_1, X) \leq \Lambda(\phi_3, X)$, so the condition on Variable Frequency is satisfied.

Putting everything together, $\preceq_{TGPay}$ is both transitive and reflective, hence it is a preorder. $\square$

Proof of Lemma 17. First we consider the case when X is an atomic argument. Since σ_1 and σ_2 are substitutions over atomic argument, by Lemma 15,

$$\sigma_1\phi \preceq_{TGPay} \phi \tag{C.1}$$

$$\phi \preceq_{TGPay} \sigma_2\phi \tag{C.2}$$

By transitivity of the preorder function,

$$\sigma_1\phi \preceq_{TGPay} \sigma_2\phi \tag{C.3}$$

So monotonicity holds when X is atomic. Next we consider the case when X is a non-atomic argument.

After substitutions, we have that:

$$\Pi(\sigma_1\phi) = \Pi(\phi) - \Lambda(\phi, X) + \Lambda(\phi, X) \times \Pi(T_1) \tag{C.4}$$

$$\Pi(\sigma_2\phi) = \Pi(\phi) - \Lambda(\phi, X) + \Lambda(\phi, X) \times \Pi(T_2) \tag{C.5}$$

Since $\Pi(T_1) \leq \Pi(T_2)$, it follows that $\Pi(\sigma_1\phi) \leq \Pi(\sigma_2\phi)$.

Let V be an arbitrary non-atomic variable from T_1, let Φ be the set of distinct non-atomic variables appeared in ϕ. To prove the condition on Variable Frequency holds we consider two cases.

1. If variable V is X or $V \notin \Phi$, then

$$\Lambda(\sigma_1\phi, V) = \Lambda(\phi, X) \times \Lambda(T_1, V) \tag{C.6}$$

$$\Lambda(\sigma_2\phi, V) = \Lambda(\phi, X) \times \Lambda(T_2, V) \tag{C.7}$$

   clearly we have,

$$\Lambda(\sigma_1\phi, V) \leq \Lambda(\sigma_2\phi, V) \tag{C.8}$$

2. Otherwise if $V \in \Phi$, then

$$\Lambda(\sigma_1\phi, V) = \Lambda(\phi, V) + \Lambda(\phi, X) \times \Lambda(T_1, V) \tag{C.9}$$

$$\Lambda(\sigma_2\phi, V) = \Lambda(\phi, V) + \Lambda(\phi, X) \times \Lambda(T_2, V) \tag{C.10}$$

clearly we still have that $\Lambda(\sigma_1\phi, V) \leq \Lambda(\sigma_2\phi, V)$.

□

Proof of Lemma 18. First, if X is an atomic argument, then replacing it with a constant or with another variable name does not affect sizes. Now suppose that X is non-atomic. By definition of $\preceq_{TGPay}$, $\Pi(\phi_1) \leq \Pi(\phi_2)$. After substitutions, we have:

$$\Pi(\sigma\phi_1) = \Pi(\phi_1) - \Lambda(\phi_1, X) + \Lambda(\phi_1, X) \times \Pi(T) \tag{C.11}$$

$$\Pi(\sigma\phi_2) = \Pi(\phi_2) - \Lambda(\phi_2, X) + \Lambda(\phi_2, X) \times \Pi(T) \tag{C.12}$$

Since $\Lambda(\phi_1, X) \leq \Lambda(\phi_2, X)$ and $\Pi(\phi_1) \leq \Pi(\phi_2)$ by Definition 78, we have that,

$$\Pi(\sigma\phi_1) \leq \Pi(\sigma\phi_2) \tag{C.13}$$

So the condition on Alphabetic Count is satisfied. We now move on to the condition of Variable Frequency.

Consider an arbitrary non-atomic variable V from T. If V does not appear in ϕ_1, and since $\Lambda(\phi_1, X) \leq \Lambda(\phi_2, X)$, we have that $\Lambda(V, \sigma_1\phi) \leq \Lambda(V, \sigma_2\phi)$. If V appears in ϕ_1 and since $\Lambda(V, \phi_1) \leq \Lambda(V, \phi_2)$, we still have the desired conclusion. This concludes the entire proof for substitution preservation. □

Proof of Lemma 19. See [Kin99] Claim 3.1. □

Proof of Theorem 11. By Lemma 16 $\preceq_{TGPay}$ is a preorder. By Lemmas 17, 18 and 19, $\preceq_{TGPay}$ satisfies Monotonicity, Substitution Preservation and Finiteness properties from Definition 11. □

List of Symbols

$A \cup B$	Append B to set A
I^*	A Dolev-Yao intruder principal
$\Lambda(\phi, X)$	Number of occurrences of variable X in ϕ
$\Pi(\phi)$	Total number of predicates, functions and non-atomic variables appearing in formula ϕ, excluding atomic variables.
Ψ^*	X.509, signature part
Ψ^T	X.509, plain-text part
Ψ	X.509 certificate
Θ^*	Time-stamp of expiration
Θ	Time-stamp of construction
$\aleph$	A variable for trustworthy principal
$\sqsupset(X)$	Is able to prove
χ	Certificate non-revocation
δ	Identity
$\emptyset$	Empty set
$\eta(X)$	Message integrity

$\gamma_K(K, P, Q)$	Symmetric key K between P, Q
$\mathfrak{R}(X, P)$	P is recipient of X
$\mathfrak{S}(X, P)$	P once uttered X
$\mathfrak{S}^{-1}(X, P)$	P never uttered X
$\omega(X, K)$	HMAC of X with key K
$\pi_P(X, Q)$	Public-key encryption of X
$\pi_S(X, P, Q)$	Symmetric key encryption of X
$\pi_V(X, Q)$	Private-key encryption of X
$\psi(P, K)$	K is private-key of P
$P \unlhd X$	P received X
$P \unrhd X$	P sent X
$\sqcap(P, X)$	P can reconstruct X
$\theta(X)$	Message digest of X
$\triangle$	Is fresh or non-expiring
$\varkappa$	Nonce
$\wp(P, K)$	K is public-key of P
$X.Y$	Concatenation of X and Y
Γ^*	Theory closure
Γ^0	Set of initial assumptions
$\Gamma^\#$	Theory representation

Ω	Pre-conditions
Σ	Post-conditions
$\Theta(T_a, T_b)$	Time interval
Υ	Intruder assertions
α	Primary premise
$P \mid\equiv X$	P believes X
β	Non-primary premise
$\circ$	Composition of substitutions
$\sharp(X)$	X is fresh
$P \mapsto \star$	P is trustworthy
$\mathfrak{D}$	Protocol
$\mathfrak{P}$	Formal proof
$\models$	Logical consequence (If ... , Then ...)
$\neg$	Negation
$\preceq$	A preorder on formula size
$I_A \rightarrow I_B : X$	Message transmission.
σ	Substitution
$\varpi(X, Y)$	Y is part of X
$\vdash_W$	Validity using W-rule or instantiation
$\vdash_{GW}$	Validity using G-rule, W-rule or instantiation

$\vdash$	Validity
$\nvdash$	Invalidity