On-the-fly Model Checking of Security Protocols

by

Guoqiang LI

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Mizuhito OGAWA

School of Information Science
Japan Advanced Institute of Science and Technology

March, 2008

Abstract

Security protocol analysis has been flourishing for almost 30 years. Many formalisms have been adopted to describe security protocols, and analyzed by various automatic and semi-automatic techniques. However, analyzing security protocols is proved to be a difficult task, due to the complication of the network. Methodologies for the analysis should be carefully designed to represent each infinity factor one assumed, such as, an unbounded number of sessions one principal participates, an unbounded number of principals one principal communicates with, and an unbounded number of messages intruders and dishonest principals produce.

A dilemma has occurred when one tries to propose a methodology for security protocol analysis, dealing with the infinity factors introduced above. On the one hand, a model for security protocols should be strong enough in expressiveness to describe any possible situation that a running protocol may reach, or it may fail in detecting subtle attacks. On the other hand, analyzing a property on a model strong in expressiveness may be undecidable. Thus automatic techniques may not terminate when detecting flaws.

This thesis proposes a sound and complete model checking method to analyze various security properties under certain assumptions. That is, when flaws are not detected, the protocol is guaranteed to be secure under these assumptions. A process calculus based on a variant of Spi calculus is introduced to describe behaviors of security protocols. Deductive systems can be inserted freely in the model, to represent infinitely many messages intruders or dishonest principals generate, due to different security assumptions. A trace semantics is chosen for the calculus, so that each possible run of a security protocol can be represented explicitly by a concrete trace.

The main contributions and achievements are:

- When various security properties of security protocols are analyzed under different assumptions, infinity factors are abstracted to be finite by several techniques, so that security properties can be checked automatically by a sound and complete on-the-fly model checking under these assumptions, including, (i) secrecy and authentication properties in bounded sessions, (ii) authentication property for recursive protocols, and (iii) non-repudiation and fairness properties in bounded sessions. Among them, (ii) and (iii) are first analyzed by model checking methods.

- Protocol-independent specifications for secrecy and authentication properties are proposed. In this approach, the specifications for secrecy and authentication properties can be generated automatically from a protocol description. In comparison, other approaches, especially process calculi based approaches, manually define a security specification dependent on a given security protocol.

The methodology is implemented by Maude. By the facility of the reachability analysis in Maude (implemented as `search`), each property can be checked at the same time when a model is generated.

Key Words: Security Protocols, On-the-fly Model Checking, Secrecy, Authentication, Non-repudiation, Fairness, Recursive Protocols, Maude

Acknowledgments

I am greatly grateful to Professor Mizuhito Ogawa, my supervisor, for his guidance, wisdom and support he has provided me throughout my doctoral education. He always helps me to clarify my research issues, inspires new ideas and enhances my thinking ability. Without him, this thesis would be impossible. The work presented in this thesis in effect should be regarded as a result of collaborations with him.

I would like to thank the dissertation committee members, who gave me instructive suggestions and comments in the evaluation of the preliminary thesis version. They are Professor Takuya Katayama, Professor Kokichi Futatsugi, Associate Professor Kazuhiro Ogata from JAIST, and Professor Shoji Yuen from Nagoya University.

I wish to continue my sincere thanks to my sub-theme supervisor, Associate Professor René Vestergaard, not only for conducting me to a very interesting field and widening my view, but also for his two lectures, *Formal Reasoning*, and *Formal Game Theory*. The lectures opened a door of formal methods in front of me. I have learned quite a lot from the lectures and discussions with him.

I would like to thank Dr. Bochao Liu, for his continuous discussions with me, which make me clear in concepts and knowledge we are both interested in. I also thank him for his guide of the knowledge in type systems to me, and for his help with my English study.

I wish to continue my thanks to my colleagues in the laboratory. They are Dr. Xin Li, Dr. Nao Hirokawa, Mr. Nguyen Van Tang and Do Mrs. Thi Bich Ngoc. It is a great pleasure to work with them everyday. They also offered me great assistances for this thesis.

I wish to thank other friends in JAIST, with whom I spent a happy time here. They are Dr. Jianwen Xiang, Dr. Jin Tian, Dr. Bochao Liu, Dr. Weiqiang Kong, Mrs. Xiaoyi Chen. I would like to devote many thanks to them for inviting me to cook in their apartments, which made me feel enjoyable; devote many thanks to Mrs. Xiaoyi Chen for driving me everywhere.

I also wish to thank those who gave me comments, and checked grammar mistakes and typos for this thesis, especially Mr. Han Zhu, who almost read the full thesis, and gave me lots of comments.

I could not have come this far without the continuous support and encouragement from my parents. I would like to devote this thesis to them.

Funding. This research is supported by the 21st Century COE "Verifiable and Evolvable e-Society" of Japan Advanced Institute of Science and Technology, funded by Japanese Ministry of Education, Culture, Sports, Science and Technology.

Contents

List of Figures

Chapter 1

Introduction

1.1 A Brief Background

In the world that is strongly dependent on distributed systems, the design of secure and dependable infrastructures is a crucial task, in which designing *security protocols* [1], core of these infrastructures, has received growing attention as a challenging problem. In open networks, such as the Internet, protocols should work under worst case assumptions, namely, messages may be eavesdropped or modified by intruders, or other dishonest principals. Attacks can be conducted even without breaking cryptography that protocols adopted, but by exploiting flaws in the protocols. The history of security protocols design is full of examples, where expectedly correct protocols that had been implemented and deployed in real applications were only found flawed years later. A most well-known case is the Needham-Schroeder authentication protocol [87]. It becomes a legend in the security protocol field.

- In 1978, R. Needham and M. Schroeder proposed Needham-Schroeder symmetric key protocol (referred to as the NSSK protocol) [87]. It became the basis for many similar protocols in later years. The authors suggested an alternative protocol based on public key cryptography, which is named the Needham-Schroeder public key protocol (referred to as the NSPK protocol).

- In 1981, Denning and Sacco demonstrated that the Needham-Schroeder symmetric Key Protocol was flawed and proposed a refined protocol [46].

- In 1994, Martín Abadi demonstrated that the protocol that Denning and Sacco refined was also flawed [9].

- In 1995, Lowe found an attack on the NSPK protocol. It is 17 years after its publication [79, 80].

Let's adopt the NSSK protocol as the first example [87] to generally understand security protocols and attacks upon them. In the NSSK protocol, principal A and principal B want to generate a session key through a key-generate server, by which they can communicate privately with each other.

[1] For a brief introduction of security protocols, please refer to Appendix A.

In the first flow, A submits a request to the server S by sending its name, its expected destination name, and a fresh nonce N_A for the later validation.

$$A \longrightarrow S : \qquad A, B, N_A$$

The server S then responds A by sending an encrypted message encrypted with the long-term key shared by A and S. The contents of the encrypted message include the nonce N_A that S received, the name of A's expected destination principal, a fresh key K_{AB}, and another encrypted message $\{K_{AB}, A\}_{K_{BS}}$, encrypted by the long-term key shared by B and S.

$$S \longrightarrow A : \qquad \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$

When A received this message, and decrypted it, it will first validate wether N_A is equal to the nonce it generated. After validation, it will forward the encrypted message $\{K_{AB}, A\}_{K_{BS}}$ to B.

$$A \longrightarrow B : \qquad \{K_{AB}, A\}_{K_{BS}}$$

Then B will generate a fresh nonce, use the key K_{AB} it received to encrypt the nonce, and send it to A.

$$B \longrightarrow A : \qquad \{N_B\}_{K_{AB}}$$

After A decrypted the message it received with the key K_{AB}, it will send $\{N_B - 1\}_{K_{AB}}$ back to B.

$$A \longrightarrow B : \qquad \{N_B - 1\}_{K_{AB}}$$

After B validates the message it received, it then makes an agreement with A to K_{AB}, and use it for the later private communications.

We know that nonces are used to identify a fresh session. But in the NSSK protocol, N_A does not work. We can see messages in the last three flows can be "reused" by intruders. That is, when A and B communicates once, and the messages are recorded by an intruder I in the network, then I can trigger B at any time. Even worse, when K_{AB} is compromised and discarded, I can make B reuse K_{AB} by sending $\{K_{AB}, A\}_{K_{BS}}$ it recorded, as follows,

$$I(A) \longrightarrow B : \qquad \{K_{AB}, A\}_{K_{BS}}$$

Then B will follows the rules of the protocol, responding as follows,

$$B \longrightarrow I(A) : \qquad \{N'_B\}_{K_{AB}}$$

Since I knows K_{AB}, it can respond as follows,

$$I(A) \longrightarrow B : \qquad \{N'_B - 1\}_{K_{AB}}$$

Then B will reuse K_{AB} to protect confidential messages for later communications. This attack is firstly reported in by Denning and Sacco [46].

The attack for the NSPK protocol is more ingenious. Let's first introduce the NSPK protocol. The flows are as follows:

$$\begin{aligned} A &\longrightarrow B : \qquad \{A, N_A\}_{+K_B} \\ B &\longrightarrow A : \qquad \{N_A, N_B\}_{+K_A} \\ A &\longrightarrow B : \qquad \{N_B\}_{+K_B} \end{aligned}$$

Principal A and B in the NSPK protocol also want to generate a fresh session key for later private communications. Firstly, A initiates a session by sending an encrypted message to B, encrypting its name A and a nonce N_A with B's public key. Then B responds A by sending back an encrypted message, encrypting N_A it received and a fresh nonce N_B with A's public key. After A validates N_A in the received message, it sends back N_B, encrypted with B's public key. B also validates the message after it received the last flow from A. Later, they will use N_B as a session key to communicate with each other confidentially.

Principals can run a protocol concurrently in different session with different other principals, some are legitimate, while others are hostile. So we assume A may communicate with a hostile principal I.

$$A \longrightarrow I : \qquad \{A, N_A\}_{+K_I}$$

Then I will pretend to be A, initiating another session by sending the message it received from A.

$$I(A) \longrightarrow B : \qquad \{A, N_A\}_{+K_B}$$

B faithfully follows the rules of the protocol, sending the message to I. Then I forwards the message it received from B to A as a response in the first session.

$$B \longrightarrow I(A) : \qquad \{N_A, N_B\}_{+K_A}$$

$$I \longrightarrow A : \qquad \{N_A, N_B\}_{+K_A}$$

A follows the protocol by sending the last message to I. I still pretends to be A, forwarding it to B as a response in the second session.

$$A \longrightarrow I : \qquad \{N_B\}_{+K_I}$$

$$I(A) \longrightarrow B : \qquad \{N_B\}_{+K_B}$$

Now B thinks it makes an agreement with A, and uses N_B as a session key for later private communications. But I knows N_B, and thus it can know the confidential messages sent by B.

To achieve security goals, researchers defined lots of security properties, such as authentication, secrecy, non-repudiation, fairness, integrity, anonymity, and so on. However, security protocols often fail to enforce the goals claimed by their designer due to various unpredicted attacks. Hence, security protocol analysis becomes a critical task for researchers. Both security protocol design and analysis prove to be challenging problems over 30 years.

1.2 Motivations and Objectives

Due to extreme importance of security protocols in the modern world, to ensure security protocols to be more reliable, and to speed up the development of new protocols, it is important to have tools that support the activity of finding flaws in protocols, or guaranteeing correctness of protocols. Various formalisms for the specification of security protocols have been proposed based on, e.g. process calculus [7, 100], belief logic [36], higher-order logic [95, 18], first-order logic [112], linear logic [51], equational logic [67], hidden algebra [90], and tree automata [86, 75, 26]. They adopted different full-automatic

or semi-automatic techniques, such as model checking based on finite models [80, 28], resolution [22, 23, 5], and higher-order theorem proving [102, 18]. However, security protocol analysis proves to be a rather difficult task. Due to the complication of the network, models for security protocol analysis should be carefully designed to depict each infinity factor one assumed. The usual factors are,

- each principal in a protocol can initiate or respond to an unbounded number of sessions;
- each principal in one session may communicate with an unbounded number of principals;
- each intruder can produce, store, duplicate, hide, or replace an unbounded number of messages based on the messages sent in the network, following the Dolev-Yao model [48];
- each dishonest principal may disobey the prescription of the protocol, sending an unbounded number of messages it can generate.

A dilemma has occurred when one tries to propose a methodology for security protocol analysis, dealing with the infinity factors above. On the one hand, a model for security protocols should be strong enough in expressiveness to describe any possible situation that a running protocol may reach, or it may fail in detecting subtle attacks of a security protocol [88, 34, 94]. On the other hand, analyzing a property on a model strong in expressiveness may be undecidable [51]. Thus automatic techniques may not terminate when detecting flaws [107, 25].

Our choice is to propose a flexible and expressive model to represent behaviors of security protocols. Deductive systems can be inserted freely to integrate the model, to represent infinitely many messages intruders or dishonest principals generate, because of different security assumptions. Driven by different security requirements, various security properties can be analyzed in the model. The analysis method is sound and complete with respect to the analyzed property, under the assumption provided by the model. The objectives of this thesis are listed as follows,

- to design a flexible expressive framework suited for analyzing various security properties under reasonable assumptions;
- to develop a fully automatic tool for security protocols analysis, sound and complete under certain assumptions. That is, when flaws are not detected, it guarantees that a protocol is secure under these assumptions;
- to find out a method to generate a specification from a protocol description automatically, making the tool easily usable.

1.3 Achievements and Contributions

This thesis adopts model checking as the execution engine, and introduces a process calculus based on a variant of Spi calculus [7] to describe behaviors of a security protocol. Deductive systems are proposed to generate messages that intruders or dishonest principals can send [28]. The main contributions are the following points.

1.3.1 Abstraction of Infinity Factors

Model Checking Authentication and Secrecy in Bounded Sessions

When analyzing authentication and secrecy, each principal is assumed to be honest. Thus only an environmental deductive system that generates intruders' messages is inserted to the model.

A sound and complete model checking method under the restriction of bounded number of sessions is proposed to analyze authentication and secrecy properties, with the ideas that:

- Due to the introduction of a *binder*, infinitely many principals that one principal may communicate with are described finitely, while other model checking methods need to impose an upper-bound to the number of principals that a principal may communicate with [80, 28, 24].
- A *parametric semantics* is proposed, abstracting infinity factors in the original model. Each variable is instantiated only when needed. The parametric semantics enjoys sound and complete to the original model with respective to the representation. Thus properties can be checked in the parametric model. This parametric approach is similar to the *lazy intruder* in the OFMC proposed by D. Basin [24].

Model checking of Recursive Protocols

A sound and complete model checking is proposed to analyze *recursive protocols*. It makes the first step towards model checking of security protocols in an unbounded number of sessions, with the following technical details.

- *Identifiers* [84, 85] are adopted to describe recursive processes. By a restriction of occurrences of identifiers in a process, it allows to analyze protocols with one recursive procedure, which can be naturally used to represent recursive protocols.
- The *pushdown system* [105], an infinite-state systems with a finite set of control locations and an unbounded stack memory, is used to abstract the infinite model. Security properties can be checked on the pushdown system.

To the best of my knowledge, this is the first model checking method applied to recursive protocols. An attack for the protocol is first detected automatically in our framework. It is also the first infinite model checking method applied to security protocol analysis.

Model Checking Non-repudiation and Fairness in Bounded Sessions

By the introduction of another deductive system to describe ability of a dishonest principals in the model, security properties, such as non-repudiation and fairness, of *fair non-repudiation protocols* in bounded sessions can be analyzed by a similar parametric approach. To the best of my knowledge, this is also the first model checking method applied to the non-repudiation property.

1.3.2 Protocol-Independent Security Specifications

Protocol-independent specifications for secrecy and authentication properties are proposed. In this approach, the specifications for secrecy and authentication properties can be generated automatically from a protocol description. In comparison, other formal methods for security protocols, especially process calculi based approaches [80, 28, 102], manually generate a security specification dependent on a given protocol. Our choice is to scan a formal protocol description statically by a syntax scanner, then the process for a security specification, named a *probing process*, is generated automatically from a formal protocol description. There are mainly two kinds of transformations.

A declaration process insertion inserts a process with only an output action, named a *declaration process*, to a formal protocol description. It describes that a principal provides a message it received, after validating this message. Declaration process insertions are used to define authentication, non-repudiation, and fairness.

A guardian process composition composes a formal protocol description to a process with only an input action, named a *guardian process*. It can be regarded as a fresh principal inserted in the network, listening all the message leaked in the environment, and trying to find out whether any confidential message is available. Guardian process compositions are used for secrecy and fairness.

1.3.3 Implementation by Maude

Maude [40], a language and system supporting both equational and rewriting logic computation for a wide range of applications, is chosen to testify our methodology. By the facility of the reachability analysis in Maude (implemented as `search`), each property can be checked at the same time while a model is being generated. Hence it is named *on-the-fly model checking* methodology.

1.4 Thesis Organizations

The rest of the thesis is organized as follows,

- Chapter 2 introduces a process calculus and its trace semantics to describe security protocols.
- Chapter 3 adopts several running examples to illustrate how to describe a security protocol "correctly" under different assumptions.
- Chapter 4 proposes an alternative parametric semantics to the calculus, by a refinement step, it is proved that transitions in parametric semantics and trace semantics are sound and complete with respect to their representations.
- Chapter 5 chooses a model for analyzing authentication and secrecy properties of security protocols in bounded sessions by avoiding identifiers. A set of *action terms* is defined to represent these security properties. Furthermore, a type-based statical analysis is proposed to compact the parametric models.

- Chapter 6 allows the model has one recursive procedure, so that recursive protocols are naturally described. By the pushdown system, unbounded number of parametric traces are represented with a finite set of control locations and an unbounded stack memory. Thus the authentication property is checked on the pushdown system.

- Chapter 7 extends the model in Chapter 5 with a deductive system to generate infinitely many messages dishonest principals may produce. In the similar way, a finite parametric model with the same representative ability is proposed to abstract all infinities. Thus security properties, such as non-repudiation and fairness, for fair non-repudiation protocols is checked in the parametric model.

- Chapter 8 introduces automatic generations from a formal protocol description to a security specification. A probing process is transformed automatically from a formal protocol description, with the aim of representing a given security property, then the formal definition of the security property is automatically generated according to the probing process. This chapter gives two algorithms to show how to generate specifications for secrecy and authentication properties.

- Chapter 9 shows implementations of parametric models introduced in Chapter 4, 5, and 6, and the on-the-fly model checking methodologies by Maude. Besides running examples, various other security protocols are also tested as case studies. The experimental results are summarized in this chapter.

- Chapter 10 gives a survey of security protocols analysis, categorized by formalisms and verifications techniques.

- Chapter 11 concludes the thesis with a discussion of the future plans.

Chapter 2

Process Calculus and Its Trace Semantics

Since security protocols work through interactions of a number of principals in parallel that send each other messages, process calculi [61, 84, 101, 21] are therefore obvious notations for describing both the principals in the network and the composition that puts them together, like CSP [100], or the Spi calculus [7]. The latter is an extension of the *π-calculus* [85, 101].

To describe behaviors of each principal in a security protocol, we choose a process calculus based on a variant of Spi calculus, in which new syntax, the *binder*, and a new primitive that binds variables, *new*, are introduced. A variable bound by the new primitive will range over a given name set. In the process calculus, we use *identifiers* [85, 84], instead of *replications* [101], to describe infinite operation in a process.

The use of identifiers is an "old-fashion" style to define recursive processes. In the past, there was an agreement that any recursive process with identifiers can be derived by replications [85]. However, recently some researches show that identifiers actually have more expressiveness than replications [93, 14, 62]. We choose to use identifiers, since with a *sequential* restriction (inspired by CCS [84]) upon a process with recursive processes, this process can be naturally described and analyzed by the *pushdown system* (see Chapter 6).

To model a hostile network, the calculus uses environment-based communication, instead of the standard channel-based communication (see Figure 2.1). Principals exchange the messages with the environment. Following the Dolev-Yao model, the environment can store, duplicate, hide or replace messages that travel on the network. It can also operate according to the rules followed by honest principals and synthesize new messages by pairing, decryption, encryption and creation of fresh nonces and keys, or by arbitrary combinations of these operations. Thus a principal waiting for an input at a given moment may expect any of the infinitely many messages the environment can produce and send in the network. These infinitely many messages are generated by a deductive system [28].

Usually, process calculi adopt channel-based communications [84, 85, 101]: two processes communicate with each other via channels. A channel can be either public (any process can access messages from the channel), or private (only authorized processes can access messages from the channel). In π-calculus, a private channel can be distributed from a process to another process dynamically via *open* and *close* transitions. Thus it is called a *mobile calculus* [85]. An environment-based process calculus can be roughly

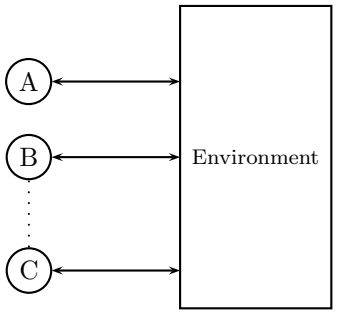


Figure 2.1: Modeling a Network by Environment-based Communication

regarded as a calculus with only a singleton public channel, through which processes communicate with each other. The channel has a memory to record every messages, and a deductive system to generate new messages. With only one channel (although more expressive), an environment-based calculus is much weaker than a channel-based calculus. However, the environment-based calculus avoids the difficulty to describe a hostile network by a specific process with intention to represent an intruder in the network [100].

To represent an unbounded number of principals with which one may communicate, we assume that a principal may send a message to any of the principals. A binder is introduced, in which a bound variable is regarded as the index of possible destinations of the message. The variable is ranged over a set of principals' names. For example,

$$(\mathtt{new}\, x : \mathcal{I})\, \overline{a1}\, \{M\}_{\mathtt{k}[A,x]}$$

represents that the principal may send the M to any possible other principals under the encryption of the encrypted key shared with the principal and the one who it intends to communicate. The usual way to restrict this kind of infinity is by bounding the number of principals in the network, so that each principal is described as only explicitly communicating with finitely many principals, including an intruder [13, 80, 28].

We choose *trace semantics* [100, 28] for the model, so that each possible run of a protocol can be represented explicitly by a *concrete trace*. Security properties will be defined on a set of traces generated by the model that describes the protocol.

2.1 Process Calculus and Concrete Trace

The syntax of our calculus is based on the Spi calculus [7]. We also introduce new syntax, binder and new.

2.1.1 Messages

Assume three countable disjoint sets: $\mathcal{L}$ for *labels*, $\mathcal{B}$ for *binder names* and $\mathcal{V}$ for *variables*. Let $a, b, c, \ldots$ indicate labels, let $\mathtt{m}, \mathtt{n}, \mathtt{k}, \ldots$ indicate binder names, and let $x, y, z, \ldots$ indicate variables.

Definition 2.1 (Messages)**.** Messages $M, N, L \ldots$ in a set $\mathcal{M}$ are defined iteratively as follows:

$pr ::=$		
	x	variable
	$\mathtt{m}[pr, \ldots, pr]$	binder
$M, N, L ::=$		
	pr	prime message
	(M, N)	pair
	$\{M\}_L$	encrypted message
	$\mathcal{H}(M)$	one-way hash message

A message is ground, if it does not contain any variables.

pr is ranged over a set of primary messages that cannot be further decomposed. A binder, $\mathtt{m}[pr_1, \ldots, pr_n]$ is a message that can be regarded as an atomic message indexed by its parameters, $pr_1, \ldots, pr_n$. For simplicity, we usually use $\mathtt{m}[\tilde{pr}]$ to denote a binder if regardless its arity. A binder with 0 arity is named a *name*, which is range over a set $\mathcal{N}$ (thus $\mathcal{N} \subset \mathcal{B}$). Let $m, n, k, \ldots$ denote names.

Remark 2.1. The first and main usage of binders is to represent infinitely many principals that a principal may communicate with, since we assume a principal who intends to send a message is supposed to send the message to anyone of the possible principals in the network, if it cannot obtain the name of the principal with which it communicates, to which it intends to send the message. An abstraction will be employed by using binders to finitely describe such an assumption. That is, the principal may send the same message to different principals, and such a sending procedure is performed only once. In essence, binders are used to parameterize keys and messages.

An alternative way to describe the communication is by using infinite process definition. By this way, a principal with intention to communicate with infinitely many principals can be described as the principal that communicates with each principal in different sessions.

There are three usages of binders in this thesis,

- to represent and parameterize encryption keys. For instance, a binder $\mathtt{k}[A, S]$ represents a symmetric key shared with principals A and S; $+\mathtt{k}[A]$ and $-\mathtt{k}[A]$ represent A's public key and private key, respectively;

- to represent an infinite set of distinguished messages with finite symbols when describe processes with recursive actions. These messages are described by nested binders. For instance, by $\mathtt{Na}[\mathtt{Null}]$, $\mathtt{Na}[\mathtt{Na}[\mathtt{Null}]]$, ..., we represent an infinite set of names in the different context;

- to represent a confidential message for the secrecy property. A confidential message will be represented as a binder when defining the secrecy property, and the parameters of the binder denote the names of the principals who shared the message. For instance, we use $\mathtt{m}[A, B]$ to represent a confidential message only shared by principal A and B.

(M, N) represents a *pair* of messages. $\{M\}_L$ is an *encrypted message* where M is its *plain message* and L is its *encryption key*. $\mathcal{H}(M)$ represents a one-way *Hash function message* where M is its parameter. We say a message M is in a message N, if M is a subterm of N.

2.1.2 Environmental Deductive System

Messages that the environment can generate are started from the current finite knowledge, denoted by S ($\subseteq \mathcal{M}$), and deduced by an *environmental deductive system*. Here, we presuppose a countable set $\mathcal{E}$ ($\subseteq \mathcal{M}$), for those ground binders, such as each principal's name, public keys, intruders' names. For example, $I, \mathtt{k}[I, S], +\mathtt{k}[A] \ldots \in \mathcal{E}$. Let $\vdash$ be the least binary relation generated by the environmental deductive system in Figure 2.2.

$$\frac{}{S \vdash M}\ M \in \mathcal{E}\ \ Env \qquad \frac{}{S \vdash M}\ M \in S\ \ Ax$$

$$\frac{S \vdash M \quad S \vdash N}{S \vdash (M, N)}\ Pair\_intro$$

$$\frac{S \vdash (M, N)}{S \vdash M}\ Pair\_elim1 \qquad \frac{S \vdash (M, N)}{S \vdash N}\ Pair\_elim2$$

$$\frac{S \vdash \{M\}_{\mathtt{k}[A,B]} \quad S \vdash \mathtt{k}[A, B]}{S \vdash M}\ Senc\_elim \qquad \frac{S \vdash M \quad S \vdash \mathtt{k}[A, B]}{S \vdash \{M\}_{\mathtt{k}[A,B]}}\ Senc\_intro$$

$$\frac{S \vdash \{M\}_{\pm\mathtt{k}[A]} \quad S \vdash \mp\mathtt{k}[A]}{S \vdash M}\ Penc\_elim \qquad \frac{S \vdash M \quad S \vdash \pm\mathtt{k}[A]}{S \vdash \{M\}_{\pm\mathtt{k}[A]}}\ Penc\_intro$$

$$\frac{S \vdash M}{S \vdash \mathcal{H}(M)}\ Hash\_intro$$

Figure 2.2: Environmental Deductive System

2.1.3 Processes

Behaviors of each principal in a security protocol are described by a process.

Definition 2.2 (Processes)**.** Let $\mathcal{P}$ be a countable set of processes which is indicated by

$P, Q, R, \ldots$. The syntax of processes is defined as follows:

$P, Q, R ::=$	
$\mathbf{0}$	Nil
$\overline{a}M.P$	output
$a(x).P$	input
$[M = N]\, P$	match
$(\texttt{new}\ x : \mathcal{A})P$	new
$(\nu\, n)P$	restriction
$let\ (x, y) = M\ in\ P$	pair splitting
$case\ M\ of\ \{x\}_L\ in\ P$	decryption
$P \| Q$	composition
$P + Q$	summation
$P; Q$	sequence
$\mathbb{A}(\tilde{pr})$	identifier

Variables x and y are bound in $a(x).P$, $(\texttt{new}\, x : \mathcal{A})P$, $let\ (x, y) = M\ in\ P$, and $case\ M\ of\ \{x\}_L\ in\ P$. Sets of free variables and bound variables in P are denoted by $f_v(P)$ and $b_v(P)$, respectively. A process P is closed if $f_v(P) = \emptyset$. A name is free in a process if it is not restricted by a restriction operator ν. Sets of free names and local names of P are denoted by $f_n(P)$ and $l_n(P)$, respectively.

Intuitively understanding,

- $\mathbf{0}$ is a *Nil* process that does nothing.
- $\overline{a}M.P$ sends message M to the environment and then behaves like P.
- $a(x).P$ awaits an input message M and behaves like $P\{M/x\}$.
- If $M = N$, $[M = N]\, P$ acts as P; otherwise it will be stuck.
- $(\texttt{new}\ x : \mathcal{A})P$ behaves like P except that x is ranged over $\mathcal{A}$ in P.
- $(\nu\, n)P$ means that the name n is local in P. Other processes will not know n before P sends it to the environment.
- If M is a pair (N, L), $let\ (x, y) = M\ in\ P$ is reduced to $P\{N/x, L/y\}$; Otherwise it will be stuck.
- Process $case\ M\ of\ \{x\}_L\ in\ P$ is reduced to $P\{N/x\}$ when M it an encrypted message $\{N\}_{L'}$ that L can decrypt; Otherwise is will be stuck.
- $P \| Q$ means that P and Q run concurrently.
- $P + Q$ behaves like P or Q.
- $P; Q$ behaves like P if P is not a Nil process; Otherwise behaves as Q.
- For any identifier $\mathbb{A}(pr_1, \ldots, pr_n)$, there must be a unique definition, that is, $\mathbb{A}(pr_1, \ldots, pr_n) \triangleq P$, where the $pr_1, \ldots, pr_n$ are distinct and are the only names and variables which occur free in P. Then $\mathbb{A}(pr'_1/pr_1, \ldots, pr'_n/pr_n)$ behaves like $P\{pr'_1/pr_1, \ldots, pr'_n/pr_n\}$. An identifier without any parameters is usually named a *constant*.

Note that labels in the input and the output processes are not channels, but tags attached to the messages of input and output actions. Furthermore, these labels are assumed to be distinct from each other.

A *recursive process* is defined by analogy with CCS [84], by proposing *process expression*. We assume a set of *schematic identifiers* [85], each having a nonnegative *arity*. In the following, X and X_i will range over schematic identifiers.

Definition 2.3 (Process Expression)**.** A *process expression* is like a process, but may contain schematic identifiers in the same way as identifers. E, F will range over agent expressions. That is, an expression is iteratively as follows,

$$\begin{aligned} E, F ::= \; & \mathbf{0} \mid \overline{a}M.E \mid a(x).E \mid [M = N]\, E \mid (\texttt{new}\ x : \mathcal{A})E \mid (\nu\, n)E \\ & \mid \mathit{let}\ (x, y) = M\ \mathit{in}\ E \mid \mathit{case}\ M\ \mathit{of}\ \{x\}_L\ \mathit{in}\ E \\ & \mid E \| F \mid E + F \mid E; F \mid X \end{aligned}$$

Definition 2.4 (Replacement [85])**.** Let X have arity n, let $\tilde{pr} = pr_1, pr_2, \ldots, pr_n$ be distinct primary messages, and assume that $f_n(P) \subseteq \{pr_1, pr_2, \ldots, pr_n\}$. The replacement of $X(\tilde{pr})$ by P in E, written $E\{X(\tilde{pr}) := P\}$, means the result of replacing each subterm $X(\tilde{pr'})$ in E by $P\{\tilde{pr'}/\tilde{pr}\}$. This extends in the obvious way to simultaneous replacement of several schematic identifiers, $E\{X_1(\tilde{pr}_1) := P_1, \ldots, X_m(\tilde{pr}_m) := P_m\}$.

It is convenient to write $E\{X_1(\tilde{pr}_1) := P_1, \ldots, X_m(\tilde{pr}_m) := P_m\}$ simply as $E(P_1, \ldots, P_m)$ or as $E(\tilde{P})$. If each P_i is $\mathbb{A}_i(\tilde{x})$, we also write $E(\mathbb{A}_1, \ldots, \mathbb{A}_m)$ or $E(\tilde{\mathbb{A}})$.

Definition 2.5 (Recursive processes)**.** A recursive process is defined as an identifier, with the following format,

$$\mathbb{A}_i(\tilde{pr}) \triangleq E(\tilde{\mathbb{A}})$$

If a process is not a recursive process, we name it a *flat process*.

Definition 2.6 (Sequential)**.** Let E be any expression. We say that a schematic identifier X is *sequential* in E if it does not occur with composition combinators in E.

This definition is inspired by the similar definition in CCS [84].

Remark 2.2 (Identifiers VS. Replications)**.** Infinite behaviors are ubiquitous in concurrent systems. Hence, they ought to be represented by processes in process calculi. Two standard processes representation of them are identifiers and replications.

To represent a recursive process by identifiers is by using expressions (see Definition 2.3) and a replacement (see Definition 2.4). It is an "old fashion" of representing infinite behaviors in process calculi [61, 84, 85].

Replications, syntactically simpler than recursive process by identifiers, take the form $!P$ [101]. Intuitively $!P$ means $P \| P \| \ldots$; an unbounded number of copies of the process.

It is common that a given process calculus, originally presented with one form of defining infinite behaviors, is later presented with the other. For example, the π-calculus was originally presented with identifiers [85], and later with replications [101]. The *Ambient calculus* was originally presented with replications [37], and later with identifiers [76].

From the above intuitive description it should be easy to see that an identifier $\mathbb{A} \triangleq \mathbb{A} \| P$ expresses the unbounded parallel behaviors of $!P$. It is less clear, however, whether replication can be used to express the unbounded behaviors of identifiers. It is through long

time that people think they have the same expressiveness, especially in π-calculus [101]. However, recently, some researches show that identifiers actually have more expressiveness than replications [93, 14, 62]. A discussed counterexample is $\mathbb{A} \triangleq (\nu\, n)(\mathbb{A}\|P)$.

We choose to use identifiers, since with a sequential restriction upon a process with recursive processes, this process can be naturally described and analyzed by the *pushdown system* (see Chapter 6).

Remark 2.3 (Channel-based communication VS. environment-based communication)**.** Usually, process calculi adopt channel-based communications [84, 85, 101]: two processes communicate with each other via channels (In π-calculus, a channel is also a name). A channel can be either public (any process can access messages from the channel), or private (only authorized processes can access messages from the channel).

In π-calculus, a private channel/name can be distributed from a process to another process dynamically via *open* and *close* transitions. Thus it is called a *mobile calculus* [85].

An environment-based calculus is similar to a calculus in which each process communicates with each other through a public "unreliable" channel. The channel has a memory to record every messages, and a deductive system to generate new messages.

An environment-based calculus is much weaker than a channel-based communication in expressiveness. Some emphases should be pointed out.

- An environment-based calculus has no synchronized operations (two processes act simultaneously). In usual process calculi, a synchronized operation is represented by an *internal action*. That is, a process with an output action at an outside channel will make a τ*-action* with a process with an input action at the same channel. For instance, in π-calculus, $\overline{a}\, m.P \mid a(x).Q \longrightarrow P \mid Q\{m/x\}$ (Here by $\mid$, we means a composition with internal actions).

- Without synchronized operations, describing an order of actions among composed processes becomes impossible. For instance, in the process in π-calculus, $(\nu\, n)(\overline{a}\, m.\ \overline{n}\, t.P \mid n(x).\overline{b}\, m'.Q)$, the private channel $\overline{n}$ can be regarded as a trigger. The action $\overline{b}\, m'$ can only be performed after $\overline{a}\, m$ is performed and trigger is precipitated. This cannot be defined in an environment-based process calculus, in which each process freely sends and receives messages from the environment.

- Computations and complex structures cannot be defined in an environment-based calculus. π-calculus has the same computation ability as the *Turing machine*. Each computation and complex structure can be defined via generating private channels and mobile computations by sending and receiving names through the privates channels. However, actions in an environment-based calculus just represent sending and receiving messages. We cannot define any complex computations on the calculus (Note that decryption and splitting are explicitly defined in our calculus by primitives). For instance, when the recursive authentication protocol is represented (see Subsection 3.4.2), the computation to generate a recursive message cannot be defined by our calculus. Thus it has to be defined separately.

2.2 Operational Semantics

Definition 2.7 (Structural congruence)**.** Structural congruence, $\equiv$, is the smallest congruence on closed processes that satisfies the axioms in Figure 2.3. Processes P and Q

are structurally congruent if $P \equiv Q$ can be inferred from the axioms listed in Figure 2.3, together with the rules of equivalence relation, that is, reflexive, symmetric, and transitive equations.

SC-COMP-ASSOC	$P\|(Q\|R)$	$\equiv$	$(P\|Q)\|R$
SC-COMP-COMM	$P\|Q$	$\equiv$	$Q\|P$
SC-COMP-INACT	$P\|\mathbf{0}$	$\equiv$	P
SC-SUM-ASSOC	$P+(Q+R)$	$\equiv$	$(P+Q)+R$
SC-SUM-COMM	$P+Q$	$\equiv$	$Q+P$
SC-RES	$(\nu m)(\nu n)P$	$\equiv$	$(\nu n)(\nu m)P$
SC-RES-INACT	$(\nu n)\mathbf{0}$	$\equiv$	$\mathbf{0}$
SC-RES-COMP	$(\nu n)(P\|Q)$	$\equiv$	$P\|(\nu n)Q \quad if\ n \notin f_n(P)$
SC-NEW	$(\texttt{new}\ x:\mathcal{A})(\texttt{new}\ y:\mathcal{B})P$	$\equiv$	$(\texttt{new}\ y:\mathcal{B})(\texttt{new}\ x:\mathcal{A})P$
SC-NEW-INACT	$(\texttt{new}\ x:\mathcal{A})\mathbf{0}$	$\equiv$	$\mathbf{0}$
SC-NEW-COMP	$(\texttt{new}\ x:\mathcal{A})(P\|Q)$	$\equiv$	$P\|(\texttt{new}\ x:\mathcal{A})Q \quad if\ x \notin f_v(P)$
SC-SEQ-INACT	$\mathbf{0};P$	$\equiv$	P

Figure 2.3: The Axioms of Structural Congruence

An *action* is a term of form $\overline{a}M$ or $a(M)$. It is ground if its attached message is ground. *Act* is defined as a set of actions. A string of ground actions represents a possible run of the protocol if each input message is deduced by messages in its prefix string. We named such a kind of string *concrete trace*, represented by s. The messages in a concrete trace s, represented by $msg(s)$, are those messages in output actions of the concrete trace s. We use $s \vdash M$ to abbreviate $msg(s) \vdash M$.

Definition 2.8 (Concrete trace and configuration)**.** A *concrete trace* s is a ground action string $s \in Act^*$ that satisfies each decomposition $s = s'.a(M).s''$ implies $s' \vdash M$. A *concrete configuration* is a pair $\langle s, P\rangle$, in which s is a concrete trace and P is a closed process.

The transition relation of concrete configurations is defined by the rules in Figure 2.4. Note that in rules $LCOM$ and $RCOM$, no reactions are provided between two composed processes, and both processes communicate with the environment.

A function `Opp` is predefined for generating complemental key in decryption and encryption. we have $\texttt{Opp}(+\texttt{k}[A]) = -\texttt{k}[A]$, $\texttt{Opp}(-\texttt{k}[A]) = +\texttt{k}[A]$ and $\texttt{Opp}(\texttt{k}[A,B]) = \texttt{k}[A,B]$. Furthermore, V is the set of free names in the source configuration. $\texttt{freshN}(V)$ is a function that generates a fresh name that does not occur in V.

For convenience, we say a concrete configuration $\langle s, P\rangle$ *reaches* $\langle s', P'\rangle$, if $\langle s, P\rangle \longrightarrow^* \langle s', P'\rangle$. A concrete configuration is a *terminated configuration* if no transition rules can be applied to it. A sequence of consecutive concrete configurations is named a *path*. A

$$
\begin{array}{rl}
(INPUT) & \langle s, a(x).P\rangle \longrightarrow \langle s.a(M), P\{M/x\}\rangle \;\; s \vdash M \\
(OUTPUT) & \langle s, \overline{a}M.P\rangle \longrightarrow \langle s.\overline{a}M, P\rangle \\
(DEC) & \langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow \langle s, P\{M/x\}\rangle \;\; L' = \mathtt{Opp}(L) \\
(PAIR) & \langle s, let\ (x,y) = (M,N)\ in\ P\rangle \longrightarrow \langle s, P\{M/x, N/y\}\rangle \\
(NEW) & \langle s, (\mathtt{new}\ x : \mathcal{A})P\rangle \longrightarrow \langle s, P\{m/x\}\rangle \quad m \in \mathcal{A} \\
(RESTRICTION) & \langle s, (\nu n)P\rangle \longrightarrow \langle s, P\{m/n\}\rangle \quad m = \mathtt{freshN}(V) \\
(MATCH) & \langle s, [M = M]P\rangle \longrightarrow \langle s, P\rangle \\
(LSUM) & \langle s, P + Q\rangle \longrightarrow \langle s, P\rangle \\
(RSUM) & \langle s, P + Q\rangle \longrightarrow \langle s, Q\rangle \\
(LCOM) & \dfrac{\langle s, P\rangle \longrightarrow \langle s', P'\rangle}{\langle s, P\|Q\rangle \longrightarrow \langle s', P'\|Q\rangle} \\
(RCOM) & \dfrac{\langle s, Q\rangle \longrightarrow \langle s', Q'\rangle}{\langle s, P\|Q\rangle \longrightarrow \langle s', P\|Q'\rangle} \\
(LSEQ) & \dfrac{\langle s, P\rangle \longrightarrow \langle s', P'\rangle}{\langle s, P; Q\rangle \longrightarrow \langle s', P'; Q\rangle} \\
(RSEQ) & \dfrac{\langle s, Q\rangle \longrightarrow \langle s', Q'\rangle}{\langle s, \mathbf{0}; Q\rangle \longrightarrow \langle s', \mathbf{0}; Q'\rangle} \\
(IND) & \dfrac{\langle s, P\{\tilde{p}\tilde{r}'/\tilde{p}\tilde{r}\}\rangle \longrightarrow \langle s', P'\rangle}{\langle s, \mathbb{A}(\tilde{p}\tilde{r}')\rangle \longrightarrow \langle s', P'\rangle} \quad \mathbb{A}(\tilde{p}\tilde{r}) \triangleq P \\
(STR) & \dfrac{P \equiv P' \quad \langle s, P'\rangle \longrightarrow \langle s', Q'\rangle \quad Q' \equiv Q}{\langle s, P\rangle \longrightarrow \langle s', Q\rangle}
\end{array}
$$

Figure 2.4: Concrete Transition Rules

concrete configuration $\langle s, P\rangle$ *generates* a concrete trace s', if $\langle s, P\rangle$ reaches $\langle s', P'\rangle$ for some P'.

Remark 2.4 (Restriction)**.** In the side-condition of $(RESTRICTION)$ rules, V is the set of free names in the source configuration. Note that $V \neq f_n(P)$. The reason is that when a composition $P\|Q$ occurs, a fresh name in P generated by $(RESTRICTION)$ rule may clash with names free occurred in Q.

This rule comes originally from [28], which is different from the traditional open and closed rules in π-calculus [101] (for semantics of π-calculus, please see Appendix B). In π-calculus, $(\nu\, n)\overline{a}n.P \mid a(x).Q \mid R$ can be reduced to $((\nu\, n)P \mid Q\{n/x\}) \mid R$. After communications between P and Q. They share the name n, which is still "untouched" by R (The scope of n is enlarged). However, in our environmental-based calculus, when a process wants to send a local message to another process. It will be "touched" by all the processes. In another word, it becomes a public and fresh name to all processes.

Remark 2.5 (Sequence)**.** By the axiom Sc-seq-inact in the structural congruence, we know that in $P; Q$, Q will be performed when P is reduced to $\mathbf{0}$, or structurally equivalent to $\mathbf{0}$. By the structural congruence, processes such as $(\mathbf{0}\|\mathbf{0}); P$ can perform the same actions that P can perform. Actually, without the sequence operator (for instance, the sub-calculi in Chapter 5, and Chapter 7), the transition rules in Figure 2.4 is enough for each process. This is the reason that we did not introduce structural congruence in [77, 78].

Furthermore, we did not consider processes in the opposite situation. That is, when P is ill-terminated, such as stuck, deadlock, or divergence. Q will never be performed. Various researches are focused in this field, proposing lots of techniques, such as *successful termination* [11], and *negative premises* [57, 27], and so on. Our calculus has a simple feature. That is, when a process is correctly terminated, it will reduce to $\mathbf{0}$, or structurally equivalent to $\mathbf{0}$. Thus, we can syntactically distinguish a correct termination from an ill one.

Remark 2.6 (Trace semantics and test semantics)**.** We adopt trace semantics, instead of traditional late/early semantics for our calculus. The reason is simple: To analyze security properties, we need to record the history a process performed. When a specific action is performed, a comparison between the message in this action and messages in the previous history will be executed. However, the traditional late/early semantics are memory-free semantics. They just consider the current actions a process can perform.

An alternative way by Abadi, et. al. is to propose test semantics (for a simple introduction of test semantics, see Subsection B.1.5 in Appendix B) for analyzing security protocols [7]. During test semantics, historical messages will be kept by the observer. Thus it can execute any comparison among messages. Abadi, et. al. adopted test equivalence to define and analyze security properties. The general idea is that a process of implementation and a process of specification of a given security protocol need to pass the same test.

Chapter 3

Representing Security Protocols

The syntax introduced in Chapter 2 is used to represent behaviors of each principal in a security protocol. However, how to describe a security protocol "correctly", so that attacks can be explored in the set of traces is not so obvious. We take the wide-mouthed frog protocol as the first example.

Example 3.1. In the wide-mouthed frog protocol, principal A shares a symmetric key $\mathtt{k}[A, S]$ with a server S; principal B shares a symmetric $\mathtt{k}[B, S]$ with S. The purpose of the protocol is to establish a new symmetric key $\mathtt{k}[A, B]$ between A and B, with which A encrypts and sends a confidential datum M to B. The protocol flows are represented as follows.

$$\begin{aligned} A \longrightarrow S : &\quad \{K_{AB}\}_{K_{AS}} \\ S \longrightarrow B : &\quad \{K_{AB}\}_{K_{BS}} \\ A \longrightarrow B : &\quad \{M\}_{K_{AB}} \end{aligned}$$

Generally, principals A, B, and S run the protocol recursively infinitely many sessions. Thus three recursive processes are used to describe these three principals. A composition is used to compose three principals who run the protocol concurrently.

$$\begin{aligned} \mathbb{A} &\triangleq (\nu\, M)\overline{a1}\,\{\mathtt{k}[A,B]\}_{\mathtt{k}[A,S]}.\overline{a2}\,\{M\}_{\mathtt{k}[A,B]}.\mathbb{A} \\ \mathbb{B} &\triangleq b1(x).case\ x\ of\{y\}_{\mathtt{k}[B,S]}\ in\ b2(z).case\ z\ of\ \{u\}_y\ in\ \mathbb{B} \\ \mathbb{S} &\triangleq s1(x).case\ x\ of\ \{y\}_{\mathtt{k}[A,S]}\ in\ \overline{s2}\,\{y\}_{\mathtt{k}[B,S]}.\mathbb{S} \\ SYS^{WMF} &\triangleq \mathbb{A}\|\mathbb{S}\|\mathbb{B} \end{aligned}$$

However, It is impossible to define properties on the description since two principals has no explicit relations between each other. In another word, principals A and B we describe above may never communicate with each other. Another consideration is the principal one communicates with. The above description represents that A communicates with B infinitely many times. However, a principal may communicate with any possible principals in the network. This is a reason that a man-in-middle attack occurs. We will represents this by the use of binders.

For generating a fresh message, we adopted the restriction operator ν to bind a name (like $(\nu\, M)$ above). When representing protocols in bounded sessions, the number of fresh

messages that principals generate is also bounded. It is enough to use a bounded set of distinguished symbols to describe each fresh message. However, when we representing recursive protocols, the number of fresh messages is unbounded. Our choice is to encoded these unbounded number of messages by nested binders.

This chapter also gives several examples for descriptions of security protocols. These examples will be taken as running examples in later chapters.

3.1 Representing Principals

When describing a protocol, We need to emphasize some rules. Otherwise, security properties cannot be defined and analyzed properly. Firstly, the principals we described should explicitly communicate with each other in one session (Thus security properties can be defined within that session). Secondly, when a principal initiates a session, we assume that it can communicate with any of the principals in the network.

Let us consider two arbitrary principals A and B who are willing to communicate with each other, as well as with any other principals through a protocol.

A principal who intends to send a message is supposed to send the message to anyone of the possible principals in the network, if it cannot obtain the name of the principal to which it intends to send the message. An abstraction will be employed by using binders to finitely describe such an assumption. That is, the principal may send the same message to different principals, and such a sending procedure is performed only once. An alternative way to describe communications is by using infinite process definition. By this way, a principal with intention to communicate with infinitely many principals can be described as the principal that communicates with each principal in different sessions.

As a receiver, its potential sender will be fixed if we only describe one session of protocols. That is, a receiver will "think" it has received the message from a principal it has known. Such an assumption is necessary when defining security properties. Otherwise the described sender and receiver may have no connections with each other, and thus security properties between them cannot be defined. For example, if A sent a message to C, and B received a message from D, it is certain that the message B received is different from the message A sent, and thus we cannot define authentication between A and B. Similarly, other properties also cannot be defined properly.

When we define the security properties in multiple sessions, the restriction will be loosened, assuming that other than in one session, a receiver can communicate with any principal. With these assumptions, security properties can be defined generally in multiple sessions.

3.2 Freshness of Messages

In the security protocol analysis, freshness of messages is one of critical points that one should consider.

In the protocol point of view, freshness problems will include *freshness of sessions*, *freshness of principals*, *freshness of keys*, and so on. Here by freshness of sessions, we mean a principal initiates a new sessions with other principals; by freshness of principals, we mean a new principals initiates or responds to a sessions; by freshness of keys, we mean that a server or a principal generates new keys for later communications.

There are several relations between these factors. For instance, bounded number of sessions implies only bounded number of principals attending the protocol, and implies only generating bounded number of fresh keys; freshness of principals implies freshness of sessions, etc.

When modeling security protocols formally, freshness of sessions is usually captured by fresh nonces; freshness of principals is usually characterized by fresh names of principals; and freshness of keys is also usually characterized by generating fresh names. Thus all of freshness problems can be described by freshness of messages. There are several ways to handle freshness of messages. For instance, in Athena, freshness of messages is expressed at the *meta-level* in the protocol model, and it is controlled during the analysis with special checks on the origin of messages [107]. In logical approaches, the BAN logic defines a new formula for the fresh messages [36]. Other logical approach uses *universal quantification* that allows one to reason about freshness inside the logic [32].

In calculus-based security protocol analysis, freshness of messages is described by the local name restriction operator, for instance, $\nu\, n$ in our calculus. Thanks to the alpha-conversion, each fresh name is guaranteed to be distinct.

When analyzing protocols in bounded sessions, the number of nonces is thus bounded, so it is with number of principals' names, and number of generated keys. It is enough to use bounded number of distinguished symbols to represent each messages. For example, for nonces, we usually take bounded number of names, $N_A, N_B, \ldots$; for principals' names, we use $A, B, S, I, \ldots$, whose number is also bounded. Note that in our calculus, keys are described by binders, functions that relate to names of principals. Thus freshness of keys is guaranteed by freshness of principals.

When analyzing protocols in unbounded sessions, the number of messages is thus unbounded. Our choice is to encoded unbounded number of messages by iterative applying binders, say, nested binders. For instance, nonces can be described by $\mathtt{N}[null]$, $\mathtt{N}[\mathtt{N}[null]]$, $\mathtt{N}[\mathtt{N}[\mathtt{N}[null]]]$, ... Later, by applying *the pushdown system* (for the detail of encodings, please refer to Chapter 6), these nested binders is described by length of stack. In the protocol point of view, we can say freshness of sessions is captured by the pushdown system. When push the stack is performed, it means a fresh session is generated; when pop is performed, it means the protocol is returned back to the previous session.

However, such an abstraction is restrictive, since a protocol can only return to the nearest previous session captured by the nonce in the stack top. Thus the pushdown system only describes unbounded number of sessions that run sequentially or recursively. For instance, the recursive protocols we analyzed in this chapter. Generally, unbounded number of sessions of a protocol should run concurrently, which cannot be straightforwardly described by the pushdown system.

Another matter that should be pointed out is, with one stack, we can only capture one type of fresh messages. That is, if the stack represents the freshness of the messages by the context (for instance, freshness of nonces for the session context), it cannot simultaneously describe another context characterized by other type of fresh messages (for instance, freshness of names of principals for the principal context). To the best of my knowledge, several other approaches sometimes neglect the freshness of principals [80, 28]. In other words, they always assume there are only bounded principals in the network.

Different types of fresh messages usually characterize different contexts. However, it is enough to only use two stacks, one for the types of fresh messages, the other as a counter, to describe general freshness of messages. However, We know that a two-stack pushdown

system is equal to the Turing machine, and thus undecidable. Recursive protocols are special. There are two types of fresh messages, nonces and names of principals, in recursive protocols. These two types of messages coincide with each other. That is, when a fresh nonce is generated, the attending principal is also a fresh one; when the protocol returns back to the previous session by using the previous nonce, so it is with the principal. Thus by one stack we can absolutely describe all types of fresh messages. This valuable characterization of recursive protocols allows us to analyze them with the pushdown system.

As we explained in this subsection, besides recursive protocols, the pushdown system based model checking can also describe and analyze security protocols with bounded principals in sequential or recursive unbounded sessions.

3.3 Protocols in Bounded sessions

3.3.1 Protocols with Server in One Session: The AG Protocol

Security protocols that exchange messages by the *symmetric key system* (also know as shared key) usually communicate via a server, Let's consider the Abadi-Gordon protocol (referred to as the AG protocol) [7] as a running example.

Example 3.2. The informal description of the AG protocol is given flow-by-flow as follows:

$$\begin{aligned} A \longrightarrow S: &\quad A, \{B, K_{AB}\}_{K_{AS}} \\ S \longrightarrow B: &\quad \{A, K_{AB}\}_{K_{SB}} \\ A \longrightarrow B: &\quad A, \{A, M\}_{K_{AB}} \end{aligned}$$

Intuitively interpreting, the principal A wants to send a message M to B encrypted by a new key K_{AB} it generates. Firstly, it sends the K_{AB} and B's name to a server S. After the S forwards the key to B, A sends a confidential message M encrypted by the key K_{AB} to B.

When the sender A initiates the protocol, it cannot obtain any information of its corresponding receiver. Thus a binder $\mathtt{k}[A, x]$ is used to describe the symmetric key shared by A and any one of principals A can communicate with in the network. Furthermore, primitive $(\mathtt{new}\ x : \mathcal{I})$ is used to bind x to an infinite set $\mathcal{I}$, which contains the names of all principals in the network. M is a confidential message. It is represented by a local name within A, restricted by ν. Sender A is described as follows:

$$A \triangleq (\mathtt{new}\ x : \mathcal{I})(\nu\, M)\overline{a1}(A, \{x, \mathtt{k}[A, x]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,x]}).\mathbf{0}$$

The receiver B firstly accepts a message from the environment. After validates the message comes from the server, it then accepts another message, which also be validated. B is described as follows:

$$\begin{aligned} B \triangleq & b1(x).case\ x\ of\ \{x'\}_{\mathtt{k}[B,S]}\ in\ let\ (y, z) = x'\ in\ [y = A]\, b2(w).\ let\ (w', w'') = w \\ & in\ [w' = A]\ case\ w''\ of\ \{u\}_z\ in\ let\ (u', u'') = u\ in\ [u' = A]\, \mathbf{0} \end{aligned}$$

The server accepts an encrypted message from a principal. It first decrypts the message, and then sends a new encrypted message according to the principal's name in the former plain message.

$$S \triangleq s1(x).let\ (y,z) = x\ in\ case\ z\ of\ \{u\}_{\mathtt{k}[y,S]}\ in\ let\ (u',u'') = u\ in\ \overline{s2}\{y,u''\}_{\mathtt{k}[u',S]}.\mathbf{0}$$

The AG protocol can be described as a composition of A, B and S.

$$SYS^{AG} \triangleq A \| S \| B$$

3.3.2 Protocols without Server in One Session: The NSPK Protocol

The AG protocol relies on a server S to establish a key between two principals by a symmetric key system. In comparison, by an *asymmetric key system* (known as public key and private key), two principals can communicate without a server. Here, we take the well-known Needham-Schroeder public key protocol (referred to as the NSPK protocol) [87] as another running example.

Example 3.3. The informal description of the NSPK protocol is given flow-by-flow as follows, in which N_A and N_B are nonces generated by A and B, respectively.

$$\begin{aligned} A &\longrightarrow B: \quad \{A, N_A\}_{+K_B} \\ B &\longrightarrow A: \quad \{N_A, N_B\}_{+K_A} \\ A &\longrightarrow B: \quad \{N_B\}_{+K_B} \end{aligned}$$

In our calculus, the NSPK protocol is represented as follows:

$$\begin{aligned} A \triangleq & (\mathtt{new}\ x_a : \mathcal{I})(\nu\ N_A)\overline{a1}\{A, N_A\}_{+\mathtt{k}[x_a]}.a2(y_a).\ case\ y_a\ of\ \{y'_a\}_{-\mathtt{k}[A]}\ in \\ & let\ (z_a, z'_a) = y'_a\ in\ [z_a = N_A]\,\overline{a3}\{z'_a\}_{+\mathtt{k}[x_a]}.\mathbf{0} \\ B \triangleq & (\nu\ N_B)\,b1(x_b).case\ x_b\ of\ \{x'_b\}_{-\mathtt{k}[B]}\ in\ let\ (y_b, y'_b) = x'_b\ in\ [y_b = A] \\ & \overline{b2}\{y'_b, N_B\}_{+\mathtt{k}[A]}.b3(z_b).\ case\ z_b\ of\ \{u_b\}_{-\mathtt{k}[B]}\ in \\ & [u_b = N_B]\,\mathbf{0} \\ SYS^{NSPK} \triangleq & A \| B \end{aligned}$$

Note that when A sent the first message, it did not who it will communicate with. Thus a bounded variable x_a in the binder $+\mathtt{k}[x_a]$ with the range of $\mathcal{I}$ (a set of principals' names) is adopted. This is the key of the description. Without the binder, we cannot detect counterexamples of the NSPK protocol in one session.

3.3.3 Protocols in Multiple Sessions: The WL Protocol

Attacks for protocols may occur between two sessions, which is called a *man-in-middle attack*. Thus, we also need to describe protocols in multiple sessions. Here we take a version of Woo-Lam protocol (referred to as the WL protocol) [114] as the third running example.

Example 3.4. The Woo-Lam protocol is defined flow-by-flow as follows:

$$\begin{aligned} A \longrightarrow B: &\quad A \\ B \longrightarrow A: &\quad N_B \\ A \longrightarrow B: &\quad \{N_B\}_{K_{AS}} \\ B \longrightarrow S: &\quad B, \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}} \\ S \longrightarrow B: &\quad \{N_B\}_{K_{BS}} \end{aligned}$$

We will analyze two sessions of the Woo-Lam protocol. $A^{(2)}$ is composed of two concurrent sessions of a sender. We assign each session a unique set of labels.

$$A^{(2)} \triangleq \overline{a1}\, A.a2(x_a).\overline{a3}\, \{x_a\}_{\mathtt{k}[A,S]}.\mathbf{0} \| \overline{a'1}\, A.a'2(x'_a).\overline{a'3}\, \{x'_a\}_{\mathtt{k}[A,S]}.\mathbf{0}$$

$B^{(2)}$ is composed of two parallel sessions of a receiver. In one of the sessions, B intends to communicate with A, so that security properties can be focused and defined in this session. Without loss of generality, we assume that in the another session, B is willing to communicate with any principal, rather than the specific principal A.

$$\begin{aligned} B^{(2)} \triangleq & (\nu\ N_B)\, b1(x_b).[x_b = A]\, \overline{b2}\, N_B.b3(y_b).\overline{b4}\, (B, \{x_b, y_b\}_{\mathtt{k}[B,S]}).b5(z_b).case\ z_b \\ & of\ \{u_b\}_{\mathtt{k}[B,S]}\ in\ [u_b = N_B]\, \mathbf{0} \| (\nu\ N'_B) b'1(x'_b).\overline{b'2}\, N'_B.b3(y'_b). \\ & \overline{b4}\, (B, \{x'_b, y'_b\}_{\mathtt{k}[B,S]}).b5(z'_b).\ case\ z'_b\ of\ \{u'_b\}_{\mathtt{k}[B,S]}\ in\ [u'_b = N'_B]\, \mathbf{0} \end{aligned}$$

$S^{(2)}$ is just composed of two sessions of a server. We need not to distinguish sets of labels of the two session.

$$\begin{aligned} S \triangleq & s1(x_s).let\ (x'_s, x''_s) = x_s\ in\ case\ x''_s\ of\ \{y_s\}_{\mathtt{k}[x'_s,S]}\ in\ let\ (z_s, w_s) = y_s\ in \\ & case\ w_s\ of\ \{u_s\}_{\mathtt{k}[z_s,S]}\ in\ \overline{s2}\, \{u_s\}_{\mathtt{k}[x'_s,S]}.\mathbf{0} \\ S^{(2)} \triangleq & S \| S \end{aligned}$$

The two-session Woo-lam protocol is described as the composition of two-session principals.

$$SYS^{(2)} \triangleq A^{(2)} \| S^{(2)} \| B^{(2)}$$

In this example, two principals A and B act as the same roles in two sessions: A as a sender, and B as a receiver. We also can describe two sessions in which two principals act as the different roles, which may also lead to attacks.

3.4 Recursive Protocols

3.4.1 Recursive Ping-Pong Protocols

Ping-pong protocols are first investigated by D. Dolev, et. al [47, 48], as a computation model. They gave an $O(n^3)$ algorithm that determines the security of a given ping-pong

protocol in bounded sessions (of number n) [47]. The security property they considered is the secrecy property against only "passive" eavesdropper, which merely listens and tries to decrypt intercepted messages. Recently, J. Srba, et. al. proved ping-pong protocols with recursive operations, but without any active intruder, are a Turing powerful model [63, 45].

Ping-pong protocols are a class of protocols, for two principals, in which the sender applies a sequence of operators to a messages M, and sends it to the other principals; in each step, one of the principal applies a sequence of operators to the message received last, and sends it back. The set of operators include encryptions and decryptions [47]. Generally, we have the following definition.

Definition 3.1 (Recursive Ping-pong protocols [47])**.** A ping-pong protocol $P(S, R)$ is a sequence $\Gamma = (\alpha_1, \alpha_2, \ldots)$ of operator-words, such that if i is odd then $\alpha \in \Sigma_S^*$ and if it is even than $\alpha \in \Sigma_R^*$.

If a ping-pong is bounded, then the length of Γ is bounded [47]. A recursive ping-pong protocol means that the length of Γ is unbounded [63]. Note that operator sets of S and R are both bounded. In [63], they only includes encryption and decryption operators.

It is easy to describe a given ping-pong protocol. Here we will try to describe a general recursive ping-pong protocol by identifiers. Let's assume that each operator set only contains encryption and decryption operators. Firstly, we define an identifier $\mathbb{A}(x)$, who has bounded number of nondeterministic choices, such as encrypts, decrypts and sends the message. x is used to describe current messages.

$$\mathbb{A}(x) \triangleq \mathbb{A}_{enc}(x) + \mathbb{A}_{dec}(x) + \overline{a1}(x).\mathbb{A}_{rec}$$

$\mathbb{A}_{enc}(x)$ is easy to define, it just like $\mathbb{A}(x)$ handle an encrypted message $\{x\}_{\mathtt{k}}$.

$$\mathbb{A}_{enc}(x) \triangleq \mathbb{A}(\{x\}_{\mathtt{k}})$$

$\mathbb{A}_{dec}(x)$ describes that the principal decrypts the message, then behaves like $\mathbb{A}$.

$$\mathbb{A}_{enc}(x) \triangleq case\ x\ of\ \{y\}_{\mathtt{k}}\ in\ \mathbb{A}(y)$$

$\mathbb{A}_{rec}$ describes the principal receives a message, then behaves like $\mathbb{A}$.

$$\mathbb{A}_{rec} \triangleq b1(x).\mathbb{A}(x)$$

In sum,

$$\mathbb{A}(x) \triangleq \mathbb{A}(\{x\}_{\mathtt{k}}) + case\ x\ of\ \{y\}_{\mathtt{k}}\ in\ \mathbb{A}(y) + \overline{a1}(x).b1(x).\mathbb{A}(x)$$

A ping-pong protocol that satisfies Definition 3.1 is defined as follows

$$\mathbb{A}(M)$$

More precisely, if one considers a set of plain messages $\mathcal{T}$ that the sender can send [63], A ping-pong protocol can be defined as

$$(\mathtt{new}\ x : \mathcal{T})\mathbb{A}(x)$$

3.4.2 The RA Protocol

Similar to security protocols introduced in Subsection 3.3, most of security protocols consist of a certain sequence of messages exchanged by the principals, in which each principal received a message replies with a new generated message that is computed from the previous messages without the need of complex methods as iteration or recursion.

But some protocols, named *recursive protocols*, contain complex actions or data structures. For instance, there are group protocols providing services not only for two principals, but for a potentially unbounded number of principals. Such protocols often contain data structures, such as arbitrary length lists. One important example of recursive protocols is the *Internet Key Exchange* (IKE) protocol. Here, we take a simple recursive protocol as an example.

In [35], a recursive protocol named the *recursive authentication protocol* (referred to as the RA protocol) is proposed, which is further explained in [94]. This protocol operates over an arbitrarily long chain of principals, terminating with a key-generated server.

Assuming there are infinitely many principals, $A_1, \ldots, A_n, \ldots$, who intend to generate session keys between each two adjacent principals by contacting with the key-generated server only once. During the communication, Each principal has two choices, it either contacts with the key-generated server to terminate the protocol, or forwards messages and itself information to its next principal to continue the protocol.

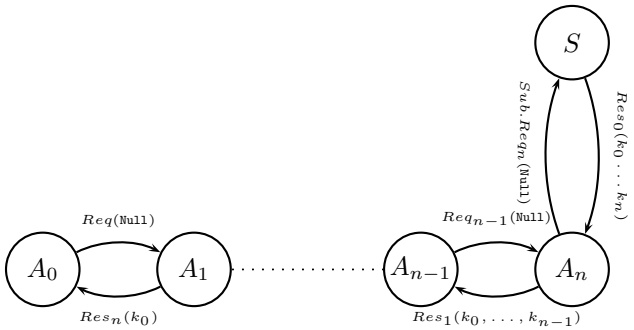


Figure 3.1: The Recursive Authentication Protocol

The protocol has three stages (see Figure 3.1).

- The first stage is the *communication stage*, each principal forwards a request that composes its information and information it accepted from the previous principal to its next principal;
- The second one is the *submission stage*, one principal stops communicating with its next principal, and submits the whole request information to the server;
- The third one is the *distribution stage*, the server recursively generates a group of session keys, and sends back to the principal who submitted the principals' information to the server. Each principal will distribute the session keys to its previous principal as a response.

For simplicity of representation, we use a convenient abbreviation of the Hash message.

$$\mathcal{H}_K(X) = (\mathcal{H}(K, X), X)$$

Example 3.5. The RA protocol is given informally as follows:

Communication Stage

$$\begin{array}{rl} A_0 \longrightarrow A_1 : & \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null}) \\ A_i \longrightarrow A_{i+1} : & \mathcal{H}_{K_{A_iS}}(A_i, A_{i+1}, N_{A_i}, X) \end{array}$$

where X is the message A_i has received from A_{i-1}. For example,

$$A_1 \longrightarrow A_2 \qquad \mathcal{H}_{K_{A_1S}}(A_1, A_2, N_{A_1}, \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null}))$$

Submission Stage

$$A_n \longrightarrow S : \mathcal{H}_{K_{A_nS}}(A_n, S, N_{A_n}, \mathcal{H}_{K_{A_{n-1}S}}(A_{n-1}, A_n, N_{A_{n-1}}, \mathcal{H}_{K_{A_{n-2}S}}(\ldots \texttt{Null}) \ldots))$$

where $\mathcal{H}_{K_{A_{n-1}S}}(\ldots \texttt{Null}) \ldots)$ is the message A_n has received from A_{n-1}.

Distribution Stage

$$\begin{array}{rl} S \longrightarrow A_n : & \{K_n, S, N_{A_n}\}_{K_{A_nS}}, \{K_{n-1}, A_{n-1}, N_{A_n}\}_{K_{A_nS}}, \\ & \{K_{n-1}, A_n, N_{A_{n-1}}\}_{K_{A_{n-1}S}}, \{K_{n-2}, A_{n-2}, N_{A_{n-1}}\}_{K_{A_{n-1}S}}, \\ & \ldots \\ & \{K_1, A_2, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_0, N_{A_1}\}_{K_{A_1S}}, \\ & \{K_0, A_1, N_{A_0}\}_{K_{A_0S}} \\ A_i \longrightarrow A_{i-1} : & \{K_{i-1}, A_i, N_{i-1}\}_{K_{A_{i-1}S}}, \{K_{i-2}, A_{i-2}, N_{i-1}\}_{K_{A_{i-1}S}}, \ldots \\ A_1 \longrightarrow A_0 : & \{K_0, A_1, N_0\}_{K_{A_0S}} \end{array}$$

When considering to describe the RA protocol, the first problem is that the number of fresh names is unbounded.

In security protocols descriptions under bounded number of sessions, messages that principals generated are also bounded, which are explicitly represented by distinguished symbols. These messages include nonces, which usually are denoted by $N_A, N_B, \ldots$; principals' names, which usually are denoted by $A, B, C, I, \ldots$.

When describing a recursive protocol, we have to represent unbounded number of messages. These messages are nonces, and principals' names and keys, which will be represented by iterative binders.

An originator (here we do not use terminologies sender and receiver, but originator and recipient, since each principal acts as a sender and a receiver simultaneously) will send a special name `Null` to show it initiates a session. Thus, its name will be represented by `A`[`Null`]; its long-term symmetric key shared with the server will be represented by `lk`[`A`[`Null`], S]; the nonce it generates will be represented by `N`[`Null`].

For the first recipient, we use $\mathtt{A[A[Null]]}$ to represent its name, $\mathtt{lk[A[A[Null]]}, S]$ to represent its long-term symmetric key shared with the server, and $\mathtt{N[N[Null]]}$ to represent the nonce it generates.

Messages of other recipients are represented similarly. Thus with finitely many binders and names, unbounded number of messages are represented.

For the simplicity of descriptions, we use several convenient abbreviations. Firstly, pair splitting is applied to input and decryption.

$$\begin{array}{rcl} a(x_1, x_2).P & \triangleq & a(x).let\ (x_1, x_2) = x\ in\ P \\ case\ M\ of\ \{x_1, x_2\}_L\ in\ P & \triangleq & case\ M\ of\ \{x\}_L\ in\ let\ (x_1, x_2) = x\ in\ P \end{array}$$

The messages of a protocol may have more than two components. We use the following standard abbreviation to generalize the syntax of pairs to arbitrary tuples, given inductively for any $k \geq 2$.

$$(M_1, M_2, \ldots, M_k, M_{k+1}) \triangleq ((M_1, M_2, \ldots, M_k), M_{k+1})$$

Similarly, we also use

$$let\ (x_1, x_2, \ldots, x_n) = M\ in\ P$$

$$a(x_1, x_2, \ldots, x_n).P$$

$$case\ M\ of\ \{x_1, x_2, \ldots, x_n\}_L\ in\ P$$

which can be straightforwardly translated into our standard syntax.

We assume there are infinitely many principals in the network, each principal can attend the protocol at most once. In a run of the protocol, the number of attended principals is unbounded, yet finite. The last principal will communicate with a server, and accept finitely many fresh keys the server generates.

We define an originator as an identifier $\mathbb{O}(x_1, x_2)$, in which x_1 and x_2 are the originator and its expected communicator's name. $\mathbb{O}(x_1, x_2)$ is defined as follows:

$$\begin{array}{rl} \mathbb{O}(x_1, x_2) \triangleq & \overline{a1}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1, x_2, \mathtt{N[Null]}, \mathtt{Null}).a2(x).case\ x\ of\ \{y_1, y_2, y_3\}_{\mathtt{lk}[x_1,S]}. \\ & [y_3 = \mathtt{N[Null]}]\,\mathbf{0} \end{array}$$

By our name-representing rules, a principal $\mathtt{A[Null]}$ which acts as an originator can be described as

$$\mathbb{O}(\mathtt{A[Null]}, \mathtt{A[A[Null]]})$$

A recipient has two choices, it can either forward the message it received and its own information to the next principal, which activate actions of its next recipient, or can submit the message to a server. It is thus described by a recursive process, an identifier $\mathbb{R}(x_1, x_2)$. Variables x_1 and x_2 in the identifier $\mathbb{R}(x_1, x_2)$ are names of its own and the previous recipient who sent it the message.

$$\begin{aligned}\mathbb{R}(x_1, x_2) \triangleq & (b1(x).let\ (y_1, y_2, y_3, y_4, y_5) = x\ in\ [y_2 = x_1] \\ & (\overline{b2}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1, \mathtt{A}[x_1], \mathtt{N}[y_3], x).\mathbb{R}(\mathtt{A}[x_1], x_1) + \\ & \overline{b3}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1, S, \mathtt{N}[y_3], x).\mathbf{0})); \\ & (b4(x).let\ (z_1, z_2, z_3) = x\ in \\ & case\ z_1\ of\ \{z_4, z_5, z_6\}_{\mathtt{lk}[x_1,S]}\ in\ [z_6 = \mathtt{N}[y_3]] \\ & case\ z_2\ of\ \{z_7, z_8, z_9\}_{\mathtt{lk}[x_1,S]}\ in\ [z_8 = x_2]\,[z_9 = \mathtt{N}[y_3]]\,\overline{b5}z_3.\mathbf{0})\end{aligned}$$

For instance, a principal $\mathtt{A}[\mathtt{A}[\mathtt{Null}]]$ which acts as an recipient can be described as

$$\mathbb{R}(\mathtt{A}[\mathtt{A}[\mathtt{Null}]], \mathtt{A}[\mathtt{Null}])$$

A server, S is represented as

$$S \triangleq s1(x).\overline{s2}\,(\mathscr{F}(x)).\mathbf{0}$$

in which $\mathscr{F} : \mathcal{M} \to \mathcal{M}$ is an iterative procedure that generates an arbitrarily long message. We name this kind of messages *recursive messages*.

$\mathscr{F}$ is defined as follows:

$$\begin{aligned}\mathscr{F}(x) = \ & \mathtt{let}\ (y_1, y_2, y_3, y_4, y_5) = x; \\ & \mathtt{let}\ t = \epsilon; \\ & \mathtt{while}\ (y_1 = \mathcal{H}(y_2, y_3, y_4, y_5, \mathtt{lk}[y_2, S])\&\&y_5! = \mathtt{Null}) \\ & \qquad \mathtt{let}\ (z_1, z_2, z_3, z_4, z_5) = y_5; \\ & \qquad \mathtt{if}\ (z_1 = \mathcal{H}(z_2, z_3, z_4, z_5, \mathtt{lk}[z_2, S])\&\&z_3 == y_2) \\ & \qquad \mathtt{then}\ t = (t, \{\mathtt{k}[y_4], y_3, y_4\}, \{\mathtt{k}[y_3], z_2, z_4\}); \\ & \qquad \mathtt{else\ raise\ error} \\ & \qquad \mathtt{endif} \\ & \qquad (y_1, y_2, y_3, y_4, y_5) := (z_1, z_2, z_3, z_4, z_5); \\ & \mathtt{endwhile} \\ & t := (t, \{\mathtt{k}[y_4], y_3, y_4\}); \\ & \mathtt{return}\ t;\end{aligned}$$

The system of the RA protocol is defined as a composition of the three processes.

$$SYS^{RA} \triangleq \mathbb{O}(\mathtt{A}[\mathtt{Null}], \mathtt{A}[\mathtt{A}[\mathtt{Null}]]) \| \mathbb{R}(\mathtt{A}[\mathtt{A}[\mathtt{Null}]], \mathtt{A}[\mathtt{Null}]) \| S$$

Chapter 4

Parametric Semantics

In the concrete model that describes a security protocol in Chapter 2, infinitely many messages the environment produces, and infinitely many principals that a principal communicates with, lead to a state explosion that makes the protocol model infinite (more precisely, infinite-branching). When applying a model checking method as an execution engine, some existing works cut down the model to a convenient finite size by imposing upper-bounds to the critical parameters, such as number of messages, number of principals, or number of pairing and encrypting operations in messages etc. [80, 13, 28].

In this chapter an alternative semantics is explored to abstract the concrete model, named *parametric semantics*, which abstracts unbounded number of branches of the concrete model's state-transition tree to be bounded.

In the concrete model, all messages in concrete traces generated by transition rules in Figure 2.4 are guaranteed to be ground, since when a concrete trace is increased by $INPUT$ $OUTPUT$ rules, each variable in the action added to the tail is substituted to a ground message. This is also the reason that the branches of the concrete model's state-transition tree are unbounded.

Concrete traces are given parametric counterparts, *parametric traces*, during parametric transitions, which may contain free variables. Each variable will only be substituted to a message (not always be ground) when they are required to satisfy some patterns. For instance, a pair, or an encrypted message. Furthermore, by a refinement step that reduces each parametric trace to a so-called *satisfiable normal form*, we can prove transitions in two semantics are sound and complete with respect to their representations.

This abstract methodology is slightly different from the traditional one, so-called *abstract interpretation* [43, 44], which is a theory of sound approximation of the semantics of a computer programs, based on *monotonic functions* over ordered sets, especially *lattices*. It can be viewed as a partial execution of a computer program which gains information about its semantics without performing all the calculations. Our methodology is to propose two independent semantics, then to prove transitions of two semantics have corresponding relations with respect to representations.

4.1 Parametric Trace and Operational Semantics

Definition 4.1 (Parametric trace and configuration)**.** A *parametric trace* $\hat{s}$ is a string of actions. A *parametric configuration* is a pair $\langle \hat{s}, P \rangle$, in which $\hat{s}$ is a parametric trace and P is a process.

The transition relation of parametric configurations is given by the rules in Figure 4.1. A substitution θ mapping from variables to messages is a *unifier* of M_1 and M_2 if $M_1\theta = M_2\theta$. A function $\texttt{Mgu}(M_1, M_2)$ returns the *most general unifier* of M_1, M_2, which is a unifier θ such that any other unifier can be written as a composition of substitution $\theta\theta'$ for some θ'.

$$(PINPUT) \quad \langle \hat{s}, a(x).P\rangle \longrightarrow_p \langle \hat{s}.a(x), P\rangle$$

$$(POUTPUT) \quad \langle \hat{s}, \overline{a}M.P\rangle \longrightarrow_p \langle \hat{s}.\overline{a}M, P\rangle$$

$$(PDEC) \quad \langle \hat{s}, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow_p \langle \hat{s}\theta, P\theta\rangle \quad \theta = \texttt{Mgu}(\{M\}_L, \{x\}_{\texttt{Opp}(L')})$$

$$(PPAIR) \quad \langle \hat{s}, let\ (x,y) = M\ in\ P\rangle \longrightarrow_p \langle \hat{s}\theta, P\theta\rangle \quad \theta = \texttt{Mgu}((x,y), M)$$

$$(PNEW) \quad \langle \hat{s}, (\texttt{new}\ x : \mathcal{A})P\rangle \longrightarrow_p \langle \hat{s}, P\{y/x\}\rangle \quad y \notin f_v(P) \cup b_v(P)$$

$$(PRESTRICTION) \quad \langle \hat{s}, (\nu n)P\rangle \longrightarrow_p \langle \hat{s}, P\{m/n\}\rangle \quad m = \texttt{freshN}(V)$$

$$(PMATCH) \quad \langle \hat{s}, [M = M']P\rangle \longrightarrow_p \langle \hat{s}\theta, P\theta\rangle \quad \theta = \texttt{Mgu}(M, M')$$

$$(PLSUM) \quad \langle \hat{s}, P + Q\rangle \longrightarrow_p \langle \hat{s}, P\rangle$$

$$(PRSUM) \quad \langle \hat{s}, P + Q\rangle \longrightarrow_p \langle \hat{s}, Q\rangle$$

$$(PLCOM) \quad \frac{\langle \hat{s}, P\rangle \longrightarrow_p \langle \hat{s}', P'\rangle}{\langle \hat{s}, P \| Q\rangle \longrightarrow_p \langle \hat{s}', P' \| Q\rangle} \quad Q' = Q\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } Q' = Q$$

$$(PRCOM) \quad \frac{\langle \hat{s}, P\rangle \longrightarrow_p \langle \hat{s}', Q'\rangle}{\langle \hat{s}, P \| Q\rangle \longrightarrow_p \langle \hat{s}', P' \| Q'\rangle} \quad P' = P\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } P' = P$$

$$(PLSEQ) \quad \frac{\langle \hat{s}, P\rangle \longrightarrow_p \langle \hat{s}', P'\rangle}{\langle \hat{s}, P; Q\rangle \longrightarrow_p \langle \hat{s}', P'; Q'\rangle} \quad Q' = Q\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } Q' = Q$$

$$(PRSEQ) \quad \frac{\langle \hat{s}, Q\rangle \longrightarrow_p \langle \hat{s}', Q'\rangle}{\langle \hat{s}, \mathbf{0}; Q\rangle \longrightarrow_p \langle \hat{s}', \mathbf{0}; Q'\rangle}$$

$$(PIND) \quad \frac{\langle \hat{s}, P\{\tilde{p}\tilde{r}'/\tilde{p}\tilde{r}\}\rangle \longrightarrow_p \langle \hat{s}', P'\rangle}{\langle \hat{s}, \mathbb{A}(\tilde{p}\tilde{r}')\rangle \longrightarrow_p \langle \hat{s}', P'\rangle} \quad \mathbb{A}(\tilde{p}\tilde{r}) \triangleq P$$

$$(PSTR) \quad \frac{P \equiv P' \quad \langle s, P'\rangle \longrightarrow_p \langle s', Q'\rangle \quad Q' \equiv Q}{\langle s, P\rangle \longrightarrow_p \langle s', Q\rangle}$$

Figure 4.1: Parametric Transition Rules

Remark 4.1 (Parametric semantics)**.** The parametric semantics above, or the *symbolic semantics* for environmental communication spi calculus introduced in [28], is a traditional semantics for process calculi [1]. The first typical characterization of this kind of semantics is that each input variable is kept un-instantiated as a placeholder, avoiding infinitely instantiation to messages. This is also a characterization of the late semantics (The concrete semantics is similar to early semantics. For simple introductions of early and late semantics for π-calculus, see Subsection B.2.1 and B.2.2 in Appendix B). Another characterization for symbolic semantics, also its difference from the late semantics, is that symbolic semantics does not perform match operations explicitly. Instead, it regards these operations as boolean formulae. A boolean formula collects the conditions on the free names of a process necessary for an action to take place (for the symbolic semantics for π-calculus, see Subsection B.2.3 in Appendix B). Our parametric semantics takes an

[1]For a simple introduction of typical semantics for process calculi, please refer to Appendix B.

alternative way. It also does not perform match (also splitting and description) explicitly. Instead, we use unification when these operations are met. Transitions may generate spurious states. Then by a refinement step, all spurious states will be gotten rid of. The parametric semantics is slightly different from the traditional symbolic semantics (although in essence they are same), and thus we adopt an alternative name.

Definition 4.2 (Concretization and abstraction)**.** Given a parametric trace $\hat{s}$, if there exists a substitution ϑ that assigns each variable to a ground message, and which satisfies $s = \hat{s}\vartheta$, where s is a concrete trace, we say that s is a *concretization* of $\hat{s}$, and $\hat{s}$ is an *abstraction* of s. ϑ is named a *concretized substitution*.

According to the definition of parametric configurations, a concrete configuration $\langle \epsilon, P \rangle$ is also a parametric configuration. We name such a configuration an *initial configuration*. We hope that from an initial configuration, each concrete trace generated by the concrete transition rules in Figure 2.4, has an abstraction generated by the parametric transition rules in Figure 4.1, and that each parametric trace has at least one concretization. Thus a bisimulation relation can be defined between them.

However, although each concrete trace does have an abstraction, some parametric traces may have no concretizations. Fortunately, parametric traces can still cover its concrete traces. That is, if a parametric trace has a concretization, then the concretization is a concrete trace generated by concrete transition rules. Otherwise the parametric trace cannot be instantiated to any concrete trace. Here we have the soundness and completeness theorem.

Theorem 4.1 (Soundness and completeness)**.** *Let $\langle \epsilon, P \rangle$ be an initial configuration, and let s be a concrete trace. $\langle \epsilon, P \rangle$ generates s, if and only if there exists $\hat{s}$, such that $\langle \epsilon, P \rangle \longrightarrow_p^* \langle \hat{s}, P' \rangle$ for some P', and s is a concretization of $\hat{s}$.*

Proof. "⇒": By an induction on the number of transitions $\longrightarrow$ and $\longrightarrow_p$, the proof is trivial in the zero-step. We assume in the n-th step the property holds. That is, for each trace s gained in the n-th $\longrightarrow$ step, there exists an $\hat{s}$ obtained by the n-th $\longrightarrow_p$ step, and $\hat{s}\vartheta = s$ holds for a substitution ϑ from variables to ground messages. Now, we perform a case analysis on the $n + 1$ step:

1. Case $\langle s, 0 \rangle$: Obviously.

2. Case $\langle s, a(x).P \rangle$: If $\langle s, a(x).P \rangle \longrightarrow \langle s.a(M), P\{M/x\} \rangle$, where M is a ground message, then we have $\langle \hat{s}, a(x).P' \rangle \longrightarrow_p \langle \hat{s}.a(x), P' \rangle$ and $s.a(M) = \hat{s}.a(x)(\vartheta \cup \{M/x\})$, where $P'\vartheta = P$. Thus $s.a(M)$ is a concretization of $\hat{s}.a(x)$, and $s.a(M) = (\hat{s}.a(x))(\vartheta \cup \{M/x\})$.

3. Case $\langle s, \overline{a}M.P \rangle$: If $\langle s, \overline{a}M.P \rangle \longrightarrow \langle s.\overline{a}M, P \rangle$, then we have $\langle \hat{s}, \overline{a}M'.P' \rangle \longrightarrow_p \langle \hat{s}.\overline{a}M', P' \rangle$, where $M'\vartheta = M$ and $P'\vartheta = P$. Since each variable in M' is already in the domain of ϑ, $(\hat{s}.\overline{a}\,M')\vartheta = s.\overline{a}\,M$, and thus $s.\overline{a}\,M$ is a concretization of $\hat{s}.\overline{a}\,M'$.

4. Case $\langle s, let\ (x, y) = (M, N)\ in\ P \rangle$: We have $\langle s, let\ (x, y) = (M, N)\ in\ P \rangle \longrightarrow \langle s, P\{M/x, N/y\} \rangle$, and (M, N) is a ground message. The counterpart configuration is $\langle \hat{s}, let\ (x, y) = M'\ in\ P' \rangle$, where $M'\vartheta = (M, N)$ and $P'\vartheta = P$. Thus $\texttt{Mgu}((x, y), M')$ will succeed and return a substitution θ, which satisfies $(x, y)\theta = M'\theta$. So $\langle \hat{s}, let\ (x, y) = M'\ in\ P' \rangle \longrightarrow_p \langle \hat{s}\theta, P'\theta \rangle$. For each variable $x_1, \ldots, x_n$ both

in domain θ and ϑ, we apply $\mathtt{Mgu}(\theta(x_i), \vartheta(x_i))$, which will return ground substitutions $\theta_1, \ldots, \theta_n$. Thus we have $s = (\hat{s}\theta)(\vartheta \backslash \{x_1, \ldots x_n\} \cup \theta_1, \cup \ldots \cup \theta_n)$.

5. Case $\langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle$: We have $\langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow \langle s, P\{M/x\}\rangle$, and M is a ground message. The counterpart configuration is $\langle \hat{s}, case\ M'\ of\ \{x\}_{L'}\ in\ P'\rangle$, where $M'\vartheta = \{M\}_{\mathtt{Opp}(L)}$, and $P'\vartheta = P$. Thus $\mathtt{Mgu}(\{x\}_{\mathtt{Opp}(L)}, M')$ will succeed and return a substitution θ, which satisfies $\{x\}_{\mathtt{Opp}(L)}\theta = M'\theta$. So $\langle \hat{s}, case\ M'\ of\ \{x\}_{L'}\ in\ P'\rangle \longrightarrow_p \langle \hat{s}\theta, P'\theta\rangle$. For each variable $x_1, \ldots, x_n$ both in domain θ and ϑ, we apply $\mathtt{Mgu}(\theta(x_i), \vartheta(x_i))$, which will return ground substitutions $\theta_1, \ldots, \theta_n$. Thus we have $s = (\hat{s}\theta)(\vartheta \backslash \{x_1, \ldots x_n\} \cup \theta_1, \cup \ldots \cup \theta_n)$.

6. Case $\langle s, [M = M]P\rangle$: If $\langle s, [M = M]P\rangle \longrightarrow \langle s, P\rangle$ and its counterpart configuration is $\langle \hat{s}, [M' = M'']P'\rangle$, where $P'\vartheta = P$, then $M'\vartheta = M''\vartheta = M$. Thus if $\theta = \mathtt{Mgu}(M', M'')$, then $\theta \subseteq \vartheta$ since the θ is the most general unifier of M' and M'' and ϑ is a unifier of them. So we have $s\vartheta = (\hat{s}\theta)\vartheta$.

7. Case $\langle s, (\mathtt{new}\ x : \mathcal{A})P\rangle$: Then we have $\langle s, (\mathtt{new}\ x : \mathcal{A})P\rangle \longrightarrow \langle s, P\{m/x\}\rangle$ while $m \in \mathcal{A}$. Its counterpart configuration is $\langle \hat{s}, (\mathtt{new}\ x)P'\rangle$ where $P'\vartheta = P$ and $s = \hat{s}(\vartheta \cup \{m/x\})$.

8. Other cases are obvious.

"$\Leftarrow$": By an induction on the number of transitions $\longrightarrow_p$ and $\longrightarrow$, the proof is trivial in the zero-step. We assume in the n-th step the property holds. That is, for each parametric trace $\hat{s}$ gained by the n-th $\longrightarrow_p$ step, if there exists a substitution ϑ from variables to ground messages, and a trace s that satisfies $s = \hat{s}\vartheta$, then s can be obtained by the n-th step of $\longrightarrow$. Now, we perform a case analysis on the $n + 1$ step:

1. Case $\langle \hat{s}, 0\rangle$: obviously.

2. Case $\langle \hat{s}, a(x).P\rangle$: If there exists a step in which $\langle \hat{s}, a(x).P\rangle \longrightarrow_p \langle \hat{s}.a(x), P\rangle$, and a ground substitution ϑ where $\hat{s}\vartheta$ is a trace, then $x\vartheta$ is a ground message which can be deduced by $s\vartheta$. So $\langle s, a(x).P'\rangle \longrightarrow \langle s.a(x\vartheta), P'\{\vartheta(x)/x\}\rangle$, where $P' = P\vartheta$.

3. Case $\langle \hat{s}, \overline{a}M.P\rangle$: If there exists a step in which $\langle \hat{s}, \overline{a}M.P\rangle \longrightarrow_p \langle \hat{s}.\overline{a}M, P\rangle$, and a ground substitution ϑ where $\hat{s}\vartheta$ is a concrete trace. So $\langle \hat{s}\vartheta, \overline{a}M\vartheta.P\vartheta\rangle \longrightarrow \langle (\hat{s}.\overline{a}M)\vartheta, P\vartheta\rangle$.

4. Case $\langle \hat{s}, let\ (x, y) = M\ in\ P\rangle$: We have $\langle \hat{s}, let\ (x, y) = M\ in\ P\rangle \longrightarrow_p \langle \hat{s}\theta, P\theta\rangle$ where $\theta = \mathtt{Mgu}((x, y), M)$, and a ground substitution ϑ where $\hat{s}\theta\vartheta$ is a concrete trace. Thus $M\theta\vartheta$ is a ground pair message. Suppose it is described by (M', N'). So $\langle \hat{s}\theta\vartheta, let\ (x, y) = (M', N')\ in\ (P\theta\vartheta)\rangle \longrightarrow \langle \hat{s}\theta\vartheta, (P\theta\vartheta)\{M'/x, N'/y\}\rangle$.

5. Case $\langle \hat{s}, case\ \{x\}_L\ of\ M\ in\ P\rangle$: We have $\langle \hat{s}, case\ \{x\}_L\ of\ M\ in\ P\rangle \longrightarrow_p \langle \hat{s}\theta, P\theta\rangle$ where $\theta = \mathtt{Mgu}(\{x\}_{\mathtt{Opp}(L)}, M)$, and a ground substitution ϑ where $\hat{s}\theta\vartheta$ is a concrete trace. Thus $M\theta\vartheta$ is a ground encrypted message. Suppose it is described by $\{M'\}_{\mathtt{Opp}(L)}$. So $\langle \hat{s}\theta\vartheta, case\ \{x\}_L\ of\ \{M'\}_{\mathtt{Opp}(L)}\ in\ (P\theta\vartheta)\rangle \longrightarrow \langle \hat{s}\theta\vartheta, (P\theta\vartheta)\{M'/x\}\rangle$.

6. Case $\langle \hat{s}, [M = M']P\rangle$: We have $\langle \hat{s}, [M = M']P\rangle \longrightarrow_p \langle \hat{s}\theta, P\theta\rangle$ where $\theta = \mathtt{Mgu}(M, M')$, and a ground substitution ϑ where $\hat{s}\theta\vartheta$ is a trace. Thus $M\theta\vartheta = M'\theta\vartheta$, and both are ground messages. So we have $\langle \hat{s}\theta\vartheta, [M = M']P\theta\vartheta\rangle \longrightarrow \langle \hat{s}\theta\vartheta, P\theta\vartheta\rangle$.

7. Case $\langle \hat{s}, (\texttt{new}\ x : \mathcal{A})P\rangle$: If there exists a step in which $\langle \hat{s}, (\texttt{new}\ x : \mathcal{A})P\rangle \longrightarrow_p \langle \hat{s}, P\rangle$, and a ground substitution ϑ where $\hat{s}\vartheta$ is a concrete trace, then $x\vartheta \in \mathcal{A}$. So $\langle s, (\texttt{new}\ x : \mathcal{A})P\rangle \longrightarrow \langle s, P\{\vartheta(x)/x\}\rangle$, where $P' = P\vartheta$.

8. Other cases are obvious.

□

4.2 Refinement Step

Theorem 4.1 shows that each concrete trace generated by an initial configuration has an abstraction. However, a parametric trace may or may not have concretizations. Let's take the following example to illustrate the reason that a parametric trace may have no concretization.

Example 4.1. Consider a naive protocol:

$$A \longrightarrow B : \{A, M\}_{K_{AB}}$$

There exists a parametric trace $b1(\{A, x\}_{\texttt{k}[A,B]})$ generated by the parametric transitions. In its concrete transitions, since $\texttt{k}[A, B]$ was not leaked in the environment, before A or B sends an encrypted message protected by $\texttt{k}[A, B]$, B cannot accept any message encrypted by $\texttt{k}[A, B]$. Thus, the parametric trace $b1(\{A, x\}_{\texttt{k}[A,B]})$ has no concretizations.

We name a message like $\{A, x\}_{\texttt{k}[A,B]}$ a *rigid message*. Intuitively, a rigid message is the pattern of a requirement of an input action. This requirement can only be satisfied by the messages generated by a proper principal. If there are no appropriate messages to satisfy the requirement, then the parametric trace has no concretizations.

A rigid message has the following definition.

Definition 4.3 (Rigid message)**.** Given a parametric trace $\hat{s} = \hat{s}'.a(M).\hat{s}''$, $\{N\}_L \in M$ is a rigid message if the following conditions are satisfied,

- L is a ground binder, and there exists a binder, or a rigid message in N;
- If L is a symmetric key, then $\hat{s}' \not\vdash L$ and $\hat{s}' \not\vdash \{N\}_L$;
- If L is a private key, then there exists some rigid message, or at least one binder in N cannot be deduced by the $\hat{s}'$, and $\hat{s}' \not\vdash \{N\}_{\texttt{Opp}(L)}$;
- If L is a public key, then $\hat{s}' \not\vdash \texttt{Opp}(L)$ and $\hat{s}' \not\vdash \{L\}_{\texttt{Opp}(L)}$.

Remark 4.2. Some researchers also regard an encrypted message where a variable is encrypted by a shared key as a rigid message [28]. For example, to represent a protocol through which A sends to B an encrypted message, $\{M\}_{\texttt{k}[A,B]}$. One of the parametric traces will be $\overline{a1}\{M\}_{\texttt{k}[A,B]}.b1(\{x\}_{\texttt{k}[A,B]})$. It seems x can only be substituted by M, and thus $\{x\}_{\texttt{k}[A,B]}$ is a rigid message. However, the communicated messages are nothing but bit streams in the network. In such a case, any bit stream with the same length as $\{M\}_{\texttt{k}[A,B]}$ can fake the message, since without comparing plain messages with other messages it already knew, B cannot distinguish whether the plain message is meaningful after decrypting the message it received. So in our definition, $\{x\}_{\texttt{k}[A,B]}$ is not a rigid message.

A parametric trace with a rigid message needs to be further substituted by trying to unify the rigid message to the atomic messages in output actions of its prefix parametric trace. We name these messages *elementary messages*, and use $el(\hat{s})$ to represent the set of elementary messages in $\hat{s}$.

Definition 4.4 (Elementary messages)**.** Let $\mathcal{U}$ be a set of messages. $dec(\mathcal{U})$ is a minimal set that satisfies

- $\mathcal{U} \subseteq dec(\mathcal{U})$;
- If $(M, N) \in dec(\mathcal{U})$, then $M, N \in dec(\mathcal{U})$;
- If $\{M\}_L \in dec(\mathcal{U})$, L is ground, and $L \in dec(\mathcal{U})$, then $M, L \in dec(\mathcal{U})$;
- If $\{M\}_L \in dec(\mathcal{U})$, and L is not ground, then $M \in dec(\mathcal{U})$.

Given a parametric trace $\hat{s}$, let $msg(\hat{s})$ be the set of all parametric messages in output actions of $\hat{s}$, then $el(\hat{s})$ is the set of minimal terms with respect to the subterm relation in $dec(msg(\hat{s}))$.

Given a parametric trace $\hat{s}$ and a message N, we say N is $\hat{\rho}$-*unifiable* in $\hat{s}$, if there exists $N' \in el(\hat{s})$ such that $\hat{\rho} = \mathtt{Mgu}(N, N')$.

The refinement step can be represented as a deductive relation, which gradually instantiates each rigid message by unifications.

Definition 4.5 (Deductive relation)**.** Let $\hat{s}$ be a parametric trace, satisfying $\hat{s} = \hat{s}_1.a(M).\hat{s}_2$, if there exists a rigid message N in M such that $N \notin el(\hat{s}_1)$, and N is $\hat{\rho}$-unifiable in $\hat{s}_1$, then $\hat{s} \rightsquigarrow \hat{s}\hat{\rho}$.

For two parametric traces $\hat{s}$ and $\hat{s}'$, if $\hat{s} \rightsquigarrow^* \hat{s}'$ and there is no $\hat{s}''$ that satisfies $\hat{s}' \rightsquigarrow \hat{s}''$, we name $\hat{s}'$ the *normal form* of $\hat{s}$. The set of normal forms of $\hat{s}$ is denoted by $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$.

Example 4.2. One of the parametric traces generated by the Abadi-Gordon protocol described in Subsection 3.3.1 is as follows. By the deductive relation, it has deduced to a normal form.

$$
\begin{aligned}
&\overline{a1}(A, \{x_1, \mathtt{k}[A, x_1]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,x_1]}).s1(x, \{y, z\}_{\mathtt{k}[x,S]}).\overline{s2}\{x, z\}_{\mathtt{k}[y,S]}.\\
&b1(\{A, t_1\}_{\mathtt{k}[B,S]}).b2(A, \{A, w_1''\}_{t_1})\\
&\rightsquigarrow\\
&\overline{a1}(A, \{x_1, \mathtt{k}[A, x_1]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,x_1]}).s1(A, \{B, z\}_{\mathtt{k}[A,S]}).\overline{s2}\{A, z\}_{\mathtt{k}[B,S]}.\\
&b1(\{A, z\}_{\mathtt{k}[B,S]}).b2(A, \{A, w_1''\}_z)\\
&\rightsquigarrow\\
&\overline{a1}(A, \{B, \mathtt{k}[A, B]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,B]}).s1(A, \{B, \mathtt{k}[A, B]\}_{\mathtt{k}[A,S]}).\\
&\overline{s2}\{A, \mathtt{k}[A, B]\}_{\mathtt{k}[B,S]}.b1(\{A, \mathtt{k}[A, B]\}_{\mathtt{k}[B,S]}).b2(A, \{A, w_1''\}_{\mathtt{k}[A,B]})\\
&\rightsquigarrow\\
&\overline{a1}(A, \{B, \mathtt{k}[A, B]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,B]}).s1(A, \{B, \mathtt{k}[A, B]\}_{\mathtt{k}[A,S]}).\\
&\overline{s2}\{A, \mathtt{k}[A, B]\}_{\mathtt{k}[B,S]}.b1(\{A, \mathtt{k}[A, B]\}_{\mathtt{k}[B,S]}).b2(A, \{A, M\}_{\mathtt{k}[A,B]})
\end{aligned}
$$

A concretization of a parametric trace $\hat{s}$ is still the concretization of $\hat{s}'$ if $\hat{s} \rightsquigarrow \hat{s}'$. Thus whether a parametric trace has concretizations is equivalent to whether there exist parametric traces in its $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$ that have concretizations.

Lemma 4.2. *If $\hat{s}$ is a parametric trace, and s is a concretization satisfying $s = \hat{s}\vartheta$ where ϑ is a concretized substitution, then $\hat{s}$ is either a normal form, or there exists $\hat{s}'$ such that $\hat{s} \rightsquigarrow \hat{s}'$ with $\hat{s}\vartheta = \hat{s}'\vartheta$.*

Proof. Let $\hat{s} = \hat{s}'.a(M).\hat{s}''$. If $\hat{s}$ is not a normal form, there exists some rigid message $\{N\}_L$ in M, such that $\{N\}_L \notin el(\hat{s}')$. Since $s = \hat{s}\vartheta$ and s is a trace, and thus $\hat{s}'\vartheta \vdash M\vartheta$, then $\{N\}_L\vartheta \in el(\hat{s}'\vartheta)$. By the definition of a rigid message, $L \notin el(\hat{s}')$, and thus $L\vartheta \notin el(\hat{s}')\vartheta$. Since $\{N\}_L\vartheta \in el(\hat{s}'\vartheta) = el(\hat{s}')\vartheta$, there exists $\{N'\}_L \in el(\hat{s}')$ such that $\{N\}_L\vartheta = \{N'\}_L\vartheta$. Thus $\{N\}_L$ and $\{N'\}_L$ are unifiable. Let $\hat{\rho} = \mathtt{Uni}(\{N\}_L, \{N'\}_L)$, then $\hat{s} \rightsquigarrow \hat{s}\hat{\rho}$. Since $\{N\}_L\vartheta = \{N'\}_L\vartheta$, each corresponding variable in two messages will be assigned to the same ground message. Thus, $\hat{s}\vartheta = \hat{s}\hat{\rho}\vartheta$. □

Lemma 4.3. *Let $\hat{s}$ be a parametric trace, and $\hat{s}'$ be a normal form in $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$. $\hat{s}'$ has a concretization, if and only if, for each decomposition $\hat{s}' = \hat{s}'_1.a(M).\hat{s}'_2$, each rigid message in M satisfies $N \in el(\hat{s}'_1)$.*

Proof. "⇒": Prove by contradiction. Assume a normal form $\hat{s}'$ has concretizations s such that $s = \hat{s}'\vartheta$. If $\hat{s}'$ does not satisfy the requirement, then there exists at least one rigid message $\{N\}_L$ in $\hat{s}'$ that $\{N\}_L \notin el(\hat{s}'_1)$. Thus $\{N\}_L\vartheta \notin el(\hat{s}'_1)\vartheta$. By definition of a rigid message, $\hat{s}'_1\vartheta \nvdash L$, then $\hat{s}'_1\vartheta \nvdash \{N\}_L\vartheta$. This contradicts the definition of a trace.

"⇐": Since the first occurrence of a variable is in an input action, let ϑ be an arbitrary concretized ground substitution that assigns each variable in $\hat{s}'$ to a name in $\mathcal{E}$, then for each decomposition $\hat{s}'\vartheta = \hat{s}'_1\vartheta.a(M\vartheta).\hat{s}'_2\vartheta$, $\hat{s}'_1\vartheta \vdash M\vartheta$ is satisfiable. Thus $\hat{s}'\vartheta$ is a trace, and also a concretization of $\hat{s}'$. □

A satisfiable normal form is a normal form of $\hat{s}$ that satisfies the requirements in Lemma 4.3.

Thus, a parametric trace has a concretization if and only if $\mathtt{snf}_{\rightsquigarrow}(\hat{s}) \neq \emptyset$.

Lemma 4.4. *Let $\hat{s}$ be a parametric trace, and let s be a trace. s is a concretization of $\hat{s}$ if and only if s is a concretization of some $\hat{s}'$ with $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow}(\hat{s})$.*

Proof. "⇒" If s is a concretization of $\hat{s}$, then there exists a concretized substitution ϑ with $s = \hat{s}\vartheta$. By Lemma 4.2 we can get either $\hat{s}$ is a normal form or $\hat{s}$ can be deduce to a parametric trace $\hat{s}'$ by $\rightsquigarrow$ such that $s = \hat{s}'\vartheta$. If $\hat{s}$ is a normal form and it has a concretization s, so $\hat{s}$ is also a satisfiable normal form according to Lemma 4.3 . If $\hat{s}$ is not a normal form, the number of rigid messages in $\hat{s}$ is finite, so $\hat{s}\vartheta = \hat{s}'\vartheta$, where $\hat{s}'$ is a normal form, by repeatedly applying lemma 4.2. Since $\hat{s}'$ has the concretization s, $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow}(\hat{s})$.

"⇐" If s is a concretization of the satisfiable normal form $\hat{s}'$ such that $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow}(\hat{s})$, we have $s = \hat{s}'\vartheta$ for some concretized substitution ϑ. $\hat{s}'$ is a normal form of $\hat{s}$, so $\hat{s}' = \hat{s}\hat{\rho}$ for some $\hat{\rho}$, in which $s = \hat{s}'\vartheta = \hat{s}\hat{\rho}\vartheta$. Thus s is a concretization of $\hat{s}$. □

Theorem 4.5. *A parametric trace $\hat{s}$ has a concretization if and only if $\mathtt{snf}_{\rightsquigarrow}(\hat{s}) \neq \emptyset$.*

The theorem is a corollary of Lemma 4.4.

Chapter 5

Secrecy and Authentication in Bounded Sessions

This chapter shows how to perform a sound and complete model checking method on security properties, such as secrecy and authentication, for a security protocol in bounded sessions. Models for protocols in bounded sessions are obtained by restricting identifiers in Definition 2.2. Then sequences become redundant, because $P;Q$ can be represented by a process in which each occurrence $\mathbf{0}$ in P is replaced by Q. Furthermore, we also avoid summations (although do not affect the method), since each principal has no nondeterministic choices when analyzing authentication and secrecy properties.

Identifiers are used to describe recursive processes. Thus without identifiers, each transition in the model will terminate lastly, and each concrete trace generated by transitions has bounded length. Then all concrete traces in this model will be abstracted to a finite set of parametric traces by the parametric semantics in Chapter 3. Therefore, model checking can be applied on the finite set of parametric traces.

In order to represent a security property, Some processes that describe behaviors for this property representation need to be inserted in the description of a security protocol. These behaviors do not belong to the prescription of a protocol. This chapter gives so-called the *specification* for a given property manually. In Chapter 7, we will discuss how to automatically generate a specification from a protocol description.

A set of *action terms* is defined in this chapter to represent security properties. We also prove that the properties defined by these action terms in the concrete model can be checked in its corresponding parametric model.

In the last subsection, a type-based statical analysis is proposed, so that set of parametric traces in a given parametric model is further reduced. This provides a more efficient way for model checking of security properties.

5.1 Sub-Calculus for Bounded Sessions

We only adopt the following primitives to describe security protocols in bounded sessions.

Definition 5.1 (Processes for bounded sessions)**.** Let $\mathcal{P}$ be a countable set of processes which is indicated by $P, Q, R, \ldots$. The syntax of processes, avoiding the identifier, the

sequence, and the summation, is defined as follows:

$$
\begin{array}{ll}
P, Q, R ::= & \\
\quad \mathbf{0} & \text{Nil} \\
\quad \overline{a}M.P & \text{output} \\
\quad a(x).P & \text{input} \\
\quad [M = N]\, P & \text{match} \\
\quad (\texttt{new}\ x : \mathcal{A})P & \text{new} \\
\quad (\nu\, n)P & \text{restriction} \\
\quad let\ (x, y) = M\ in\ P & \text{pair splitting} \\
\quad case\ M\ of\ \{x\}_L\ in\ P & \text{decryption} \\
\quad P \| Q & \text{composition}
\end{array}
$$

The concrete semantics is also restricted, which is shown in Figure 5.1.

$$
\begin{array}{rl}
(INPUT) & \langle s, a(x).P\rangle \longrightarrow \langle s.a(M), P\{M/x\}\rangle \ \ s \vdash M \\
(OUTPUT) & \langle s, \overline{a}M.P\rangle \longrightarrow \langle s.\overline{a}M, P\rangle \\
(DEC) & \langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow \langle s, P\{M/x\}\rangle \ \ L' = \texttt{Opp}(L) \\
(PAIR) & \langle s, let\ (x, y) = (M, N)\ in\ P\rangle \longrightarrow \langle s, P\{M/x, N/y\}\rangle \\
(NEW) & \langle s, (\texttt{new}\ x : \mathcal{A})P\rangle \longrightarrow \langle s, P\{m/x\}\rangle \quad m \in \mathcal{A} \\
(RESTRICTION) & \langle s, (\nu n)P\rangle \longrightarrow \langle s, P\{m/n\}\rangle \quad m = \texttt{freshN}(V) \\
(MATCH) & \langle s, [M = M]P\rangle \longrightarrow \langle s, P\rangle \\
(LCOM) & \dfrac{\langle s, P\rangle \longrightarrow \langle s', P'\rangle}{\langle s, P\|Q\rangle \longrightarrow \langle s', P'\|Q\rangle} \\
(RCOM) & \dfrac{\langle s, Q\rangle \longrightarrow \langle s', Q'\rangle}{\langle s, P\|Q\rangle \longrightarrow \langle s', P\|Q'\rangle} \\
(STR) & \dfrac{P \equiv P' \quad \langle s, P'\rangle \longrightarrow \langle s', Q'\rangle \quad Q' \equiv Q}{\langle s, P\rangle \longrightarrow \langle s', Q\rangle}
\end{array}
$$

Figure 5.1: Concrete Transition Rules for Bounded Sessions

Any transition in Figure 5.1 will terminate lastly. Thus the only reason make the state-transition tree of the model infinite is its infinite-branching, which is caused by the rules $INPUT$ and NEW. The following are two examples to show the infinity factors.

Example 5.1. Let's take the following two examples,

- For the $INPUT$ rules, a process is defined as

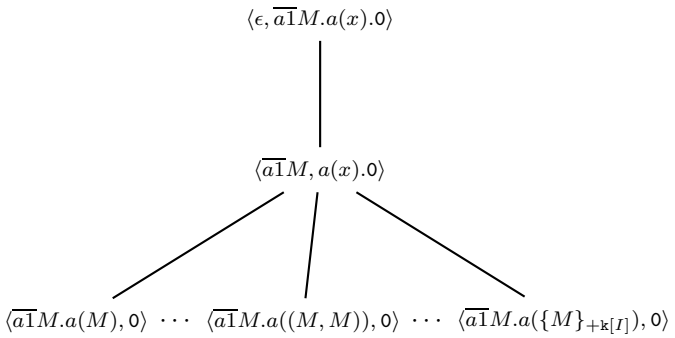
$$A \triangleq \overline{a1}M.a2(x).\mathbf{0}$$

(see its state-transition tree in (a), Figure 5.2). $\langle \epsilon, A\rangle$ will transit to $\langle \overline{a1}M, a2(x).\mathbf{0}\rangle$ by the $OUTPUT$ rule, which can then transit to infinitely many concrete configurations. The reason is that the concrete trace $\overline{a1}M$ can deduce infinitely many messages due to the environmental deductive system.

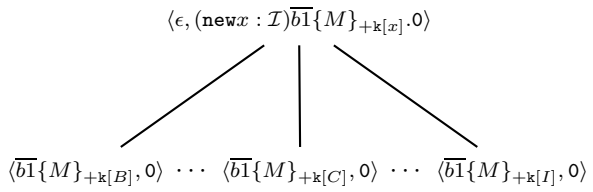
- For the NEW rules, a process is defined as

$$B \triangleq (\texttt{new}\ x : \mathcal{I})\overline{b1}\{M\}_{+\mathtt{k}[x]}$$

(see its state-transition tree in (b), Figure 5.2), where $\mathcal{I}$ is an infinite set of principals' names. According to the NEW rule, x will be instantiated to any name in $\mathcal{I}$, which thereafter leads to infinitely many concrete traces by $OUTPUT$ rules.



(a) $INPUT$ rule



(b) NEW rule

Figure 5.2: State-Transition Trees for Infinite Systems

5.2 Action Terms

A set of action terms is defined to represent security properties.

Definition 5.2. Let α range over the set of actions. The set of action terms is defined as follows:

$$\begin{aligned} T ::= &\ \alpha \mid \neg T \mid T \wedge T \mid T \vee T \\ \sigma ::= &\ T \mid T \leftarrow T \mid T \hookrightarrow_F T \end{aligned}$$

Action terms inductively defined by T are *state action terms*, and those defined by σ are *path action terms*. A state action term is also a path action term.

We define two relations: the relation $\models_t$ between a concrete trace and a state action term, and $\models$ between a concrete configuration and a path action term. Two relations are inductively defined as follows:

- $s \models_t \alpha$: there exists a ground substitution ρ from variables to ground messages such that $\alpha\rho$ occurs in s.
- $s \models_t \neg T$: $s \not\models_t T$.
- $s \models_t T_1 \wedge T_2$: $s \models_t T_1$ and $s \models_t T_2$.
- $s \models_t T_1 \vee T_2$: $s \models_t T_1$ or $s \models_t T_2$.
- $\langle s, P\rangle \models T$: for each concrete trace s' generated by $\langle s, P\rangle$, $s' \models_t T$ holds.
- $\langle s, P\rangle \models T_1 \hookleftarrow T_2$: for each concrete trace s' generated by $\langle s, P\rangle$, if there is a ground substitution ρ such that $s' \models_t T_2\rho$, then $s' \models_t T_1\rho$, and $T_1\rho$ occurs before $T_2\rho$ in s'.
- $\langle s, P\rangle \models T_1 \hookrightarrow_F T_2$: for each concrete configuration $\langle s', P'\rangle$ reached by $\langle s, P\rangle$, if there is a ground substitution ρ such that $s' \models_t T_1\rho$, then for every path starting from $\langle s', P'\rangle$, there exists a concrete trace s'' such that $s'' \models_t T_2\rho$.

5.3 Representing Security Properties

5.3.1 Security Properties for the AG Protocol

Secrecy

The secrecy property intuitively means that the environment should never learn a confidential datum the principals exchange. For example, in the AG protocol described in Subsection 3.3.1, a confidential datum is M, which should be guaranteed never leaked in the environment without any protection.

We assume that a sender may send a message to any possible principal if it cannot gain the information about its destination from previous messages. Under this assumption, we cannot confirm whether the message is sent to the specific receiver we represented. In order to define the secrecy property, a binder will be used, instead of a name to represent a confidential datum M in the AG protocol. Principals who are expected to know the datum are denoted explicitly by parameters of the binder. For instance, By $\mathtt{M}[A, B]$, we means that the datum is only shared by A and B. With this modification, the action labeled with $a2$ in the description of the AG protocol (see Subsection 3.3.1) should be modified to $\overline{a2}(A, \{A, \mathtt{M}[A, x]\}_{\mathtt{k}[A,x]})$. Suppose the modified system is defined as $SYS^{AG'}$.

Remark 5.1. Other methods [28, 80] restricted the number of principals in the network, and assumed that a principal explicitly sent a message to an intruder at the same time when it sent another message to a legitimate principal. Thus it is enough to check whether the message sent to the legitimate principal is secure or not (the message sent to the intruder is obvious not secure). However, in our methodology, we abstract two sending actions by one action using binders. Thus the message can be either sent to an intruder, or sent to a legitimate principal. So we should parameterize the message by a binder, explicitly assigning the principals who share the message as parameters of the binder, so that only the message with two concerned principals' names as its parameters is checked.

In addition, A process $check(x).\mathbf{0}$ is introduced to compose the description of the AG protocol, which represents a listener in the network, who can access any message leaked in the environment.

$$SYS_s^{AG} \triangleq SYS^{AG'} \| \mathbf{check(x).0}$$

Remark 5.2. Intuitively, $check(x).\mathbf{0}$ can be regarded as a fresh principal inserted in the network, listening all the message leaked in the network, and trying to find out whether some confidential message is leaked. So the secrecy property is defined as the fresh principal cannot access the confidential message.

The secrecy property of the AG protocol is characterized by an action term, as follows.

Characterization 5.1 (Secrecy for the AG protocol)**.** *Given the formal process for secrecy of the AG protocol, it satisfies the secrecy property, if*

$$\langle \epsilon, SYS_s^{AG} \rangle \models \neg check(\mathtt{M}[A, B])$$

Authentication

The authentication property is another important security property that has been studied in security protocol analysis. Intuitively, it means that a message that purports to be sent from a certain principal was indeed originated by that principal.

We exploit an already existing and widely used way to specify authentication properties, called *correspondence assertion*, which was first introduced by Thomas Y.C. Woo and Simon S. Lam [113].

For authentication, a process, $\overline{acc}\, M.\mathbf{0}$ will inserted to the description of the AG protocol, substituting some occurrences of $\mathbf{0}$. For the AG protocol, we are interesting in checking the authentication in the third flow (see Subsection 3.3.1). M in $\overline{acc}\, M.\mathbf{0}$ is thus the same message attached by $b2$. The process for the AG protocol SYS_a^{AG} is then transformed as follows, in which A and S are original descriptions represented in Subsection 3.3.1.

$$\begin{aligned} B_a \triangleq & b1(x).case\ x\ of\ \{x'\}_{\mathtt{k}[B,S]}\ in\ let\ (y,z) = x'\ in\ [y = A]\, b2(w).\ let\ (w', w'') = w \\ & in\ [w' = A]\ case\ w''\ of\ \{u\}_z\ in\ let\ (u', u'') = u\ in\ [u' = A]\, \overline{\mathbf{acc}}\, \mathbf{w.0} \\ SYS_a^{AG} \triangleq & A \| B_p \| S \end{aligned}$$

Remark 5.3. Intuitively, $\overline{acc}\, M.\mathbf{0}$ describes that a principal provides a message it received, after validates the message is coming from the principal it expected. In another words, it "thinks" the M it received comes from an expected principal (although in reality, it may or may not).

So the authentication property of the AG protocol is characterized formally as follows.

Characterization 5.2 (Authentication for the AG protocol)**.** *Given the process for authentication of Abadi-Gordon protocol, the sender is correctly authenticated to the receiver, if*

$$\langle \epsilon, SYS_a^{AG} \rangle \models \overline{a2}\, x \hookleftarrow \overline{acc}\, x$$

5.3.2 Security Properties for the NSPK Protocol

Secrecy

The confidential datum concerned in the NSPK protocol (see Subsection 3.3.2) is N_B. By the similar consideration, we use a binder to replace the name N_B in the description. Furthermore, the listener's process will also be inserted in the description. The process for secrecy of the NSPK protocol is given as follows, in which A is the original description in Subsection 3.3.2.

$$\begin{aligned} B_s \triangleq & (\nu\, N_B) b1(x_b).case\ x_b\ of \{x'_b\}_{-\mathtt{k}[B]}\ in\ let\ (y_b, y'_b) = x'_b\ in\, [y_b = A] \\ & \overline{b2}\{y'_b, \mathtt{N_B}[B, A]\}_{+\mathtt{k}[A]}.b3(z_b).\ case\ z_b\ of\ \{u_b\}_{-\mathtt{k}[B]}\ in \\ & [u_b = \mathtt{N_B}[B, A]]\, \mathbf{0} \\ SYS_s^{NSPK} \triangleq & A \| B_s \| \mathbf{check}(\mathbf{x}).\mathbf{0} \end{aligned}$$

The secrecy property of the NSPK protocol is characterized as follows.

Characterization 5.3 (Secrecy for the NSPK protocol)**.** *Given the formal process for secrecy of the NSPK protocol, it satisfies the secrecy property, if*

$$\langle \epsilon, SYS_s^{NSPK} \rangle \models \neg check(\mathtt{N_B}[B, A])$$

Authentication

The authenticated flow concerned in the NSPK protocol is also the third flow (see Subsection 3.3.2). M in the process $\overline{acc}\, M.\mathbf{0}$ is thus the message attached by $b3$. The process for the NSPK protocol SYS_a^{NSPK} is transformed as follows, in which A is the original description represented in Subsection 3.3.2.

$$\begin{aligned} B_a \triangleq & (\nu\, N_B)\, b1(x_b).case\ x_b\ of \{x'_b\}_{-\mathtt{k}[B]}\ in\ let\ (y_b, y'_b) = x'_b\ in\, [y_b = A] \\ & \overline{b2}\{y'_b, N_B\}_{+\mathtt{k}[A]}.b3(z_b).\ case\ z_b\ of\ \{u_b\}_{-\mathtt{k}[B]}\ in \\ & [u_b = N_B]\, \overline{\mathbf{acc}}\, \mathbf{z_b}.\mathbf{0} \\ SYS_a^{NSPK} \triangleq & A \| B_a \end{aligned}$$

So the authentication property of the NSPK protocol is characterized formally as follows.

Characterization 5.4 (Authentication for the NSPK protocol)**.** *Given the formal process for authentication of NSPK protocol, the sender is correctly authenticated to the receiver, if*

$$\langle \epsilon, SYS_a^{NSPK} \rangle \models \overline{a3}\, x \hookleftarrow \overline{acc}\, x$$

5.3.3 Security Properties for the Multiple WL Protocol

There are no confidential data to be checked in the Woo-Lam protocol. We only concern the authentication of the Woo-Lam protocol in multiple sessions (see Subsection 3.3.3). For this protocol, we are interesting in checking the authentication of the third

flow. M in the process $\overline{acc}\,M.\mathbf{0}$ will be replaced to the message attached by $b3$. $B^{(2)}$ is transformed as follows. $SYS_a^{(2)}$ is therefore the process for the authentication property of the multiple Woo-Lam protocol, in which $A^{(2)}$ and $S^{(2)}$ are original processes defined in Subsection 3.3.3..

$$\begin{aligned} B_a^{(2)} \triangleq & (\nu\, N_B)b1(x_b).[x_b = A]\,\overline{b2}\,N_B.b3(y_b).\overline{b4}\,(B,\{x_b,y_b\}_{\mathtt{k}[B,S]}).b5(z_b).case\ z_b \\ & of\ \{u_b\}_{\mathtt{k}[B,S]}\ in\ [u_b = N_B]\,\overline{\mathbf{acc}}\,\mathbf{y_b}.\mathbf{0} \| (\nu\, N'_B)b'1(x'_b).\overline{b'2}\,N'_B.b3(y'_b). \\ & \overline{b4}\,(B,\{x'_b,y'_b\}_{\mathtt{k}[B,S]}).b5(z'_b).\ case\ z'_b\ of\ \{u'_b\}_{\mathtt{k}[B,S]}\ in\ [u'_b = N'_B]\,\mathbf{0} \\ SYS_a^{(2)} \triangleq & A^{(2)} \| S^{(2)} \| B_p^{(2)} \end{aligned}$$

We define the authentication property as follows: if the label $\overline{acc}$ occurs in a trace attached to a message, then at least one label in $\overline{a3}$ and $\overline{a'3}$ attached to the same message occurs in the same trace before $\overline{acc}$.

Characterization 5.5 (Authentication for the two-session Woo-Lam protocol)**.** *Given the formal process for authentication of the two-session Woo-Lam protocol, the sender is correctly authenticated to the receiver, if*

$$\langle \epsilon, SYS_a^{(2)} \rangle \models (\overline{a3}\,x \vee \overline{a'3}x) \hookleftarrow \overline{acc}\,x$$

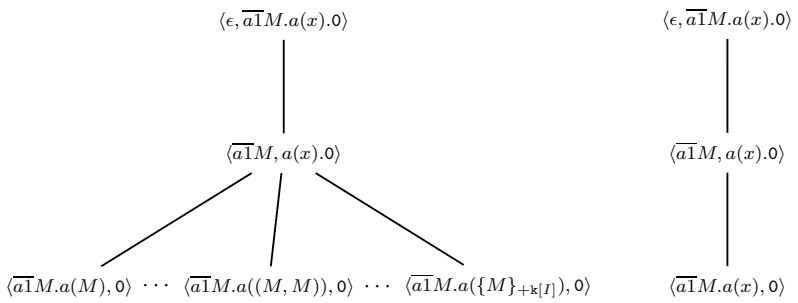
5.4 Checking Security Properties

The counterpart parametric model for protocols in bounded sessions is given in Figure 5.3.

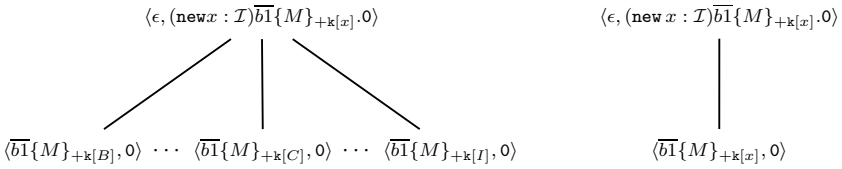
$$\begin{array}{rl}
(PINPUT) & \langle \hat{s}, a(x).P \rangle \longrightarrow_p \langle \hat{s}.a(x), P \rangle \\
(POUTPUT) & \langle \hat{s}, \overline{a}M.P \rangle \longrightarrow_p \langle \hat{s}.\overline{a}M, P \rangle \\
(PDEC) & \langle \hat{s}, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P \rangle \longrightarrow_p \langle \hat{s}\theta, P\theta \rangle \\
 & \qquad \theta = \mathtt{Mgu}(\{M\}_L, \{x\}_{\mathtt{Opp}(L')}) \\
(PPAIR) & \langle \hat{s}, let\ (x,y) = M\ in\ P \rangle \longrightarrow_p \langle \hat{s}\theta, P\theta \rangle \quad \theta = \mathtt{Mgu}((x,y), M) \\
(PNEW) & \langle \hat{s}, (\mathtt{new}\ x : \mathcal{A})P \rangle \longrightarrow_p \langle \hat{s}, P\{y/x\} \rangle \qquad y \notin f_v(P) \cup b_v(P) \\
(PRESTRICTION) & \langle \hat{s}, (\nu n)P \rangle \longrightarrow_p \langle \hat{s}, P\{m/n\} \rangle \qquad m = \mathtt{freshN}(V) \\
(PMATCH) & \langle \hat{s}, [M = M']P \rangle \longrightarrow_p \langle \hat{s}\theta, P\theta \rangle \qquad \theta = \mathtt{Mgu}(M, M') \\
(PLCOM) & \dfrac{\langle \hat{s}, P \rangle \longrightarrow_p \langle \hat{s}', P' \rangle}{\langle \hat{s}, P \| Q \rangle \longrightarrow_p \langle \hat{s}', P' \| Q \rangle} \quad Q' = Q\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } Q' = Q \\
(PRCOM) & \dfrac{\langle \hat{s}, P \rangle \longrightarrow_p \langle \hat{s}', Q' \rangle}{\langle \hat{s}, P \| Q \rangle \longrightarrow_p \langle \hat{s}', P' \| Q \rangle} \quad P' = P\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } P' = P \\
(PSTR) & \dfrac{P \equiv P' \quad \langle s, P' \rangle \longrightarrow_p \langle s', Q' \rangle \quad Q' \equiv Q}{\langle s, P \rangle \longrightarrow_p \langle s', Q \rangle}
\end{array}$$

Figure 5.3: Parametric Transition Rules for Bounded Sessions

By these parametric transitions, infinity factors of concrete transitions in Figure 5.1 will be abstracted to be finite. For instance, examples in Example 5.1 will be finitely abstracted, as showed in Figure 5.4.



(a) Abstracting $INPUT$ rule



(b) Abstracting NEW rule

Figure 5.4: State-Transition Trees for Abstracting Infinite Systems

Security properties can be checked in the parametric model, by the following two reasons: Firstly, each parametric transition in Figure 5.3 will terminate. Secondly, since each parametric trace has the bounded length. The refinement step defined in Subsection 4.2 has the following two facts. Thus the refinement step of a given parametric trace will also terminate.

Fact 5.1. *Given a parametric trace* $\hat{s}$, $\mathtt{nf}_{\rightsquigarrow}(\hat{s})$ *is finite.*

Fact 5.2. *Given a parametric trace* $\hat{s}$, $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$ *is finite.*

According to the two facts introduced above, the set of parametric traces generated by the parametric model is finite. In order to check these security properties, we need to simulate action terms definitions on its parametric traces.

Definition 5.3. Let T be a state action term, and let $\hat{s}$ be a parametric trace that has concretizations. We say $\hat{s} \models_t T$, if for each concretization s of $\hat{s}$, $s \models_t T$.

Definition 5.4. Let σ be a path action term, and let $\langle \hat{s}, P \rangle$ be a parametric configuration, where $\hat{s}$ has concretizations. We say $\langle \hat{s}, P \rangle \models \sigma$, if for each concretization s of $\hat{s}$, where $s = \hat{s}\vartheta$, $\langle \hat{s}\vartheta, P\vartheta \rangle \models \sigma$.

An action α is *$\hat{\rho}$-unifiable* in a parametric trace $\hat{s}$ if the parametric message in α can be unified to the message attached to the same label as α in $\hat{s}$, and $\hat{\rho}$ is the result of the unification.

Lemma 5.1. *Given a parametric trace* $\hat{s}$,

1. $\hat{s} \models_t \alpha$ *if and only if,* α *is* $\hat{\rho}$*-unifiable in* $\hat{s}$*, and for each satisfiable normal form in* $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho})$ *satisfying* $\hat{s}\hat{\rho}\hat{\rho}'$*,* $\alpha\hat{\rho}\hat{\rho}'$ *occurs in* $\hat{s}\hat{\rho}\hat{\rho}'$*.*

2. $\hat{s} \models_t \neg\alpha$ *if and only if* $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho}) = \emptyset$ *when* α *is* $\hat{\rho}$*-unifiable in* $\hat{s}$*.*

3. *For any state action term* T*,* $\hat{s} \models_t T$ *is decidable.*

Proof. Given a parametric trace $\hat{s}$,

1. "⇒": Prove by contradictions: Firstly, if α is not $\hat{\rho}$-unifiable in $\hat{s}$, then α cannot be $\hat{\rho}$-unifiable in all concretizations of $\hat{s}$. Thus given a concretization s in which α is not $\hat{\rho}$-unifiable, $\alpha\rho$ is not occurs in s for any ground substitution ρ. Then $s \not\models_t \alpha$. So $\hat{s} \not\models_t \alpha$. Secondly, if α is $\hat{\rho}$-unifiable in $\hat{s}$, but there exists a satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho})$ satisfying $\hat{s}\hat{\rho}\hat{\rho}'$, and $\alpha\hat{\rho}\hat{\rho}'$ does not occur in $\hat{s}\hat{\rho}\hat{\rho}'$, then for any concretization s' of $\hat{s}\hat{\rho}\hat{\rho}'$, $s' \not\models_t \alpha\hat{\rho}'$. According to Lemma 4.4, s' is also a concretization of $\hat{s}$ and thus $\hat{s} \not\models_t \alpha$.

   "⇐": If α is $\hat{\rho}$-unifiable in $\hat{s}$, and for each satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho})$ satisfying $\hat{s}\hat{\rho}\hat{\rho}'$, $\alpha\hat{\rho}\hat{\rho}'$ occurs in $\hat{s}\hat{\rho}\hat{\rho}'$, Then $\alpha\hat{\rho}\hat{\rho}'\rho$ occurs in any concretization $\hat{s}\hat{\rho}\hat{\rho}'\rho$. According to Lemma 4.4, each concretization of $\hat{s}$ is also a concretization of one satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$. Thus for all concretization s of $\hat{s}$, $s \models_t \alpha$. So $\hat{s} \models_t \alpha$.

2. "⇒": Prove by contradictions: If α is $\hat{\rho}$-unifiable in $\hat{s}$, and $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho}) \neq \emptyset$, by Theorem 4.5, $\hat{s}\hat{\rho}$ has concretizations. We choose an arbitrary concretization s that satisfies $s = \hat{s}\hat{\rho}\vartheta$, and then $\alpha\hat{\rho}\vartheta$ occurs in s. Thus $\hat{s} \not\models_t \neg\alpha$, which contradicts the assumption.

   "⇐": If α is $\hat{\rho}$-unifiable in $\hat{s}$ with $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho}) = \emptyset$, by Theorem 4.5, $\hat{s}\hat{\rho}$ has no concretization. Since a concretization of $\hat{s}$ in which α is $\hat{\rho}$-unifiable is also a concretization of $\hat{s}\hat{\rho}$, then for each concretization s of $\hat{s}$, $s \models_t \neg\alpha$. Thus $\hat{s} \models_t \neg\alpha$.

3. It is easy to show that $\hat{s} \models_t T_1 \wedge T_2$ if and only if $\hat{s} \models_t T_1$ and $\hat{s} \models_t T_2$, and $\hat{s} \models_t T_1 \vee T_2$ if and only if $\hat{s} \models_t T_1$ or $\hat{s} \models_t T_2$. Furthermore, action terms satisfy De Morgan's laws. That is, $\hat{s} \models_t \neg(T_1 \wedge T_2)$ if and only if $\hat{s} \models_t \neg T_1 \vee \neg T_2$, and $\hat{s} \models_t \neg(T_1 \vee T_2)$ if and only if $\hat{s} \models_t \neg T_1 \wedge \neg T_2$. So any state action term problem can be checked on a parametric trace, Thus $\hat{s} \models_t T$ is decidable.

□

Lemma 5.2. *Given a concrete configuration* $\langle s, P\rangle$*, and two state action terms* T_1 *and* T_2*,* $\langle s, P\rangle \models T_1 \hookrightarrow_F T_2$*, if and only if for each concrete configuration* $\langle s', P'\rangle$ *reached by* $\langle s, P\rangle$*, if there exists a ground substitution* ρ *such that* $s' \models_t T_1\rho$*, then for each terminated configuration* $\langle s'', P''\rangle$ *reached by* $\langle s', P'\rangle$*,* $T_2\rho$ *occurs in* s''*.*

Proof. "⇒": By the definition, $\langle s, P\rangle \models T_1 \hookrightarrow_F T_2$ if for each concrete configuration $\langle s', P'\rangle$ reached by $\langle s, P\rangle$, if there is a ground substitution ρ such that $s' \models_t T_1\rho$, then for every path starting from $\langle s', P'\rangle$, there exists a concrete trace s'' such that $s'' \models_t T_2\rho$. According to the concrete transition rules in Figure 5.1, if $\langle s, P\rangle$ generates a trace s', then $s' = s.s''$ for some s''. Thus, for every path starting from $\langle s', P'\rangle$, if there exists a concrete trace s'' such that $s'' \models_t T_2\rho$, then in the terminated configuration of that path $\langle s''', P'''\rangle$, $s''' \models T_2\rho$.

"⇐": The condition in the above lemma satisfies the definition of $\langle s, P\rangle \models T_1 \hookrightarrow_F T_2$. □

The above Lemma 5.2 provides an easy way to check $\hookrightarrow_F$ in parametric traces.

Theorem 5.3. *Given an initial configuration* $\langle \epsilon, P\rangle$,

1. *Given a state action term* T, $\langle \epsilon, P\rangle \models T$, *if and only if for each parametric trace* $\hat{s}$ *generated by* $\langle \epsilon, P\rangle$, $\hat{s} \models_t T$.

2. *Given two state action terms* T_1 *and* T_2, $\langle \epsilon, P\rangle \models T_2 \hookleftarrow T_1$, *if and only if for each parametric trace* $\hat{s}$ *generated by* $\langle \epsilon, P\rangle$, *if* T_1 *is* $\hat{\rho}$*-unifiable in* $\hat{s}$, *then for each satisfiable normal form in* $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho})$ *satisfying* $\hat{s}\hat{\rho}\hat{\rho}'$, $T_2\hat{\rho}\hat{\rho}'$ *occurs before* $T_1\hat{\rho}\hat{\rho}'$.

3. *Given two state action terms* T_1 *and* T_2, $\langle \epsilon, P\rangle \models T_1 \hookrightarrow_F T_2$, *if and only if for each parametric configuration* $\langle \hat{s}', P'\rangle$ *reached by* $\langle \epsilon, P\rangle$, *if* T_1 *is* $\hat{\rho}$*-unifiable in* $\hat{s}'$, *then for each terminated parametric configuration* $\langle \hat{s}''\hat{\rho}, P''\hat{\rho}\rangle$ *reached by* $\langle \hat{s}'\hat{\rho}, P'\hat{\rho}\rangle$, *either* $\hat{s}''\hat{\rho}$ *cannot deduce any satisfiable normal forms, or* $T_2\hat{\rho}\hat{\rho}'$ *occurs in each satisfiable normal form* $\hat{s}''\hat{\rho}\hat{\rho}'$ *in* $\mathtt{snf}_{\rightsquigarrow}(\hat{s}'\hat{\rho})$.

Proof. 1. It is a corollary of Theorem 4.5 and Lemma 5.1.

2. "⇒": Prove by contradictions: Given an arbitrary parametric trace $\hat{s}$ generated by $\langle \epsilon, P\rangle$, if T_1 is not $\hat{\rho}$-unifiable in $\hat{s}$, then $\hat{s} \not\models_t T_1$ according to Lemma 5.1. Thus for any concretization s of $\hat{s}$, $s \not\models_t T_1$. So $\langle \epsilon, P\rangle \not\models T_2 \hookleftarrow T_1$, which contradicts to our assumption. Otherwise, assume a satisfiable normal form $\hat{s}\hat{\rho}\hat{\rho}'$ in $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho})$, and $T_2\hat{\rho}\hat{\rho}'$ does not occur before $T_1\hat{\rho}\hat{\rho}'$ in $\hat{s}\hat{\rho}\hat{\rho}'$. Let s' be a concretization of $\hat{s}\hat{\rho}\hat{\rho}'$ satisfying $s' = \hat{s}\hat{\rho}\hat{\rho}'\vartheta$. Thus $T_2\hat{\rho}\hat{\rho}'\vartheta$ does not occur before $T_1\hat{\rho}\hat{\rho}'\vartheta$ in s', that is, $s' \not\models T_2 \hookleftarrow T_1$. s' is also the concretization of $\hat{s}$. Thus $\hat{s} \not\models T_2 \hookleftarrow T_1$, which contradicts the assumption.

   "⇐": Given an arbitrary parametric trace $\hat{s}$ generated by $\langle \epsilon, P\rangle$, if T_1 is $\hat{\rho}$-unifiable in $\hat{s}$, and for each satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s}\hat{\rho})$ satisfying $\hat{s}\hat{\rho}\hat{\rho}'$, $T_2\hat{\rho}\hat{\rho}'$ occurs before $T_1\hat{\rho}\hat{\rho}'$ in $\hat{s}\hat{\rho}\hat{\rho}'$, then for each concretization satisfying $\hat{s}\hat{\rho}\hat{\rho}'\vartheta$, $\hat{s}\hat{\rho}\hat{\rho}'\vartheta \models T_2 \hookleftarrow T_1$. By Lemma 4.4, the concretization is also a concretization of $\hat{s}$. According to Theorem 4.5, any trace in $\langle \epsilon, P\rangle$ is a concretization of some counterpart parametric trace. Thus we have $\langle \epsilon, P\rangle \models T_2 \hookleftarrow T_1$.

3. "⇒": Prove by contradiction: Given an arbitrary parametric trace $\hat{s}$ generated by $\langle \epsilon, P\rangle$, where T_1 is $\hat{\rho}$-unifiable in $\hat{s}'$, we suppose there exists a path from $\langle \hat{s}'\hat{\rho}, P'\hat{\rho}\rangle$ that will be terminated to $\langle \hat{s}''\hat{\rho}, P''\hat{\rho}\rangle$, and $T_2\hat{\rho}\hat{\rho}'$ does not occur in a satisfiable normal form $\hat{s}''\hat{\rho}\hat{\rho}'$ in $\mathtt{snf}_{\rightsquigarrow}(\hat{s}''\hat{\rho})$. Then for any ground substitution ρ, $\langle \hat{s}''\hat{\rho}\hat{\rho}'\rho, P''\hat{\rho}\hat{\rho}'\rho\rangle$ is a terminated configuration of $\langle \epsilon, P\rangle$ reached by $\langle \hat{s}'\hat{\rho}\rho, P'\hat{\rho}\rho\rangle$. $\hat{s}'\hat{\rho}\rho \models_t T_1\rho$, while $\hat{s}''\hat{\rho}\hat{\rho}'\rho \not\models_t T_2\rho$. By Lemma 5.2, $\langle \epsilon, P\rangle \not\models T_1 \hookrightarrow_F T_2$, which contradicts the assumption.

   "⇐": For each parametric trace $\hat{s}'$ generated by $\langle \epsilon, P\rangle$, if T_1 is $\hat{\rho}$-unifiable in $\hat{s}'$, then for each concretization s of $\hat{s}'$(if it has), satisfying $s = \hat{s}'\rho$, $s \models_t T_1\rho$. Furthermore, by the assumption, for each terminated parametric configuration $\langle \hat{s}''\hat{\rho}, P''\hat{\rho}\rangle$ reached by $\langle \hat{s}'\hat{\rho}, P'\hat{\rho}\rangle$, if it can reduce to some satisfiable normal form, according to Theorem 4.5, $\hat{s}'\hat{\rho}$ has concretization. For each satisfiable normal form, denoted by $\hat{s}''\hat{\rho}\hat{\rho}'$, $T_2\hat{\rho}\hat{\rho}'$

occurs after $T_1\hat{\rho}\hat{\rho}'$. Thus $\langle \hat{s}''\hat{\rho}\hat{\rho}'\rho, P''\hat{\rho}\hat{\rho}'\rho\rangle$ is a terminated configuration, and $T_2\hat{\rho}\hat{\rho}'\rho$ occurs in $\hat{s}''\hat{\rho}\hat{\rho}'\rho$. According to Lemma 5.2, we have $\langle \epsilon, P\rangle \models T_1 \hookrightarrow_F T_2$.

□

Actually, Theorem 5.3 implicitly shows the algorithm to check whether a system satisfies a path action term.

5.5 Counterexamples and Attacks

5.5.1 Results and Discussion of the AG Protocol

There are no counterexamples in the parametric model that violate both secrecy and authentication definitions of the AG protocol. However, this protocol does not satisfy a so-called *agreement authentication*[1] property, which is stronger than the authentication defined by correspondence assertion. This attack can be found by bisimulation methods [7].

5.5.2 Attacks of the NSPK Protocol and Its Modification

For the NSPK protocol, a parametric trace that does not satisfy the secrecy property is as follows,

$$\overline{a1}\{A, N_A\}_{+\mathtt{k}[x_a]}.b1(\{A, N_A\}_{-\mathtt{k}[B]}).\overline{b2}\{N_A, \mathtt{N_B}[B, A]\}_{+\mathtt{k}[A]}.a2(\{N_A, \mathtt{N_B}[B, A]\}_{-\mathtt{k}[A]}).$$
$$\overline{a3}\{\mathtt{N_B}[B, A]\}_{+\mathtt{k}[x_a]}.check(\mathtt{N_B}[B, A])$$

The counterexample shows that in $\overline{a3}$, A may send the message he intends to send to B to other principals (thus leaking the confidential message $\mathtt{N_B}[A, B]$).

Similarly, a counterexample to the authentication

$$\overline{a1}\{A, N_A\}_{+\mathtt{k}[x_a]}.b1(\{A, N_A\}_{-\mathtt{k}[B]}).\overline{b2}\{N_A, N_B\}_{+\mathtt{k}[A]}.a2(\{N_A, N_B\}_{-\mathtt{k}[A]}).$$
$$\overline{a3}\{N_B\}_{+\mathtt{k}[x_a]}.b3(\{N_B\}_{-\mathtt{k}[B]}).\overline{acc}\{N_B\}_{-\mathtt{k}[B]}$$

which means that B thinks that he accepts the message from A, while actually A can send the message to any one of possible principals.

The two counterexamples actually represent the same well-known man-in-middle attack [79, 80], which is given as:

$$\begin{aligned}
A \longrightarrow I: &\quad \{A, N_A\}_{+K_I} && \text{(a1)}\\
I(A) \longrightarrow B: &\quad \{A, N_A\}_{+K_B} && \text{(b1)}\\
B \longrightarrow I(A): &\quad \{N_A, N_B\}_{+K_A} && \text{(b2)}\\
I \longrightarrow A: &\quad \{N_A, N_B\}_{+K_A} && \text{(a2)}\\
A \longrightarrow I: &\quad \{N_B\}_{+K_I} && \text{(a3)}\\
I(A) \longrightarrow B: &\quad \{N_B\}_{+K_B} && \text{(b3)}
\end{aligned}$$

[1] See in Appendix A for more details about the property.

The fixed NSPK protocol [80] revised the second flow of the original protocol.

$$\begin{aligned} A &\longrightarrow B: \quad \{A, N_A\}_{+K_B} \\ B &\longrightarrow A: \quad \{B, N_A, N_B\}_{+K_A} \\ A &\longrightarrow B: \quad \{N_B\}_{+K_B} \end{aligned}$$

It avoids such an attack. In the second flow, A will check whether the principal whom it intends to communicate with is identical to the principal name it has received. The principal A is represented as follows.

$$\begin{aligned} A_{fix} \triangleq & (\nu\ N_A)(\texttt{new}\ x_a : \mathcal{I})\overline{a1}\{A, N_A\}_{+\texttt{k}[x_a]}.a2(y_a). \\ & \mathit{case}\ y_a\ \mathit{of}\ \{y'_a\}_{-\texttt{k}[A]}\ \mathit{in}\ \mathit{let}\ (z_a, z'_a) = y'_a\ \mathit{in}\ [z_a = x_a] \\ & \mathit{let}\ (w_a, w'_a) = z'_a\ \mathit{in}\ [w_a = N_A]\,\overline{a3}\{w'_a\}_{+\texttt{k}[z_a]}.\mathbf{0} \end{aligned}$$

With the match operation $[z_a = x_a]$, when we instantiate z_a during generating a parametric trace, x_a will be instantiated at the same time. Thus both the label $\overline{a3}$ and the label $\overline{acc}$ are attached to the same message. Following the same actions order of the counterexample for the authentication, it is not a counterexample any more.

$$\begin{aligned} & \overline{a1}\{A, N_A\}_{+\texttt{k}[B]}.b1(\{A, N_A\}_{-\texttt{k}[B]}).\overline{b2}\{B, N_A, N_B\}_{+\texttt{k}[A]}. \\ & a2(\{B, N_A, N_B\}_{-\texttt{k}[A]}).\overline{a3}\{N_B\}_{+\texttt{k}[B]}.b3(\{N_B\}_{-\texttt{k}[B]}). \\ & \overline{acc}\{N_B\}_{-\texttt{k}[B]} \end{aligned}$$

This means that A will not send the message labeled $a3$ to any possible principal, since it has accepted the communicator's name via z_a.

5.5.3 Attacks of the Multiple WL Protocol and Its Modification

For the Woo-Lam protocol in multiple sessions, the counterexamples can be found in its parametric traces. One of them is shown as follows:

$$\begin{aligned} & b1(A).b'1(x'_b).\overline{b2}\,N_B.\overline{b'2}\,N'_B.b3(y_b).b'3(\{N_B\}_{\texttt{k}[x'_b,S]}).\overline{b'4}(B, \{x'_b, \{N_B\}_{\texttt{k}[x'_b,S]}\}_{\texttt{k}[B,S]}). \\ & \overline{b4}\,(B, \{A, y_b\}_{\texttt{k}[B,S]}).s1(x_s, \{y_s, \{z_s\}_{\texttt{k}[y_s,S]}\}_{\texttt{k}[x_s,S]}).s1(B, \{x'_b, \{N_B\}_{\texttt{k}[x'_b,S]}\}_{\texttt{k}[B,S]}). \\ & \overline{s2}\,\{z_s\}_{\texttt{k}[x_s,S]}.\overline{s2}\,\{N_B\}_{\texttt{k}[B,S]}.b5(\{N_B\}_{\texttt{k}[B,S]}).\overline{acc}\,y_b \end{aligned}$$

In this counterexample, there are no actions labeled with ax. It means that an intruder can completely imitate A. It is a bit difficult to understand the counterexample, which actually represents the following attack.

$$\begin{aligned} I(A) &\longrightarrow B: \quad A && \text{(a1)} \\ B &\longrightarrow I(A): \quad N_B && \text{(a2)} \\ I &\longrightarrow B: \quad I && \text{(b1)} \\ B &\longrightarrow I: \quad N'_B && \text{(b2)} \\ I(A) &\longrightarrow B: \quad y_b && \text{(a3)} \\ B &\longrightarrow S: \quad B, \{A, y_b\}_{K_{BS}} && \text{(a4)} \\ I &\longrightarrow B: \quad \{N_B\}_{K_{IS}} && \text{(b3)} \\ B &\longrightarrow S: \quad B, \{I, \{N_B\}_{K_{IS}}\}_{K_{BS}} && \text{(b4)} \\ S &\longrightarrow B: \quad \{N_B\}_{K_{BS}} && \text{(a5(b5))} \end{aligned}$$

The reason that the attack occurs is that B cannot distinguish which session the last message belongs to. To correct the protocol, one possible solution is that the server S appends the information of B's communicated principal in the encrypted message in the last flow [55].

$$\begin{array}{ll} A \longrightarrow B: & A \\ B \longrightarrow A: & N_B \\ A \longrightarrow B: & \{N_B\}_{K_{AS}} \\ B \longrightarrow S: & B, \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}} \\ S \longrightarrow B: & \{A, N_B\}_{K_{BS}} \end{array}$$

However, the modified protocol has a replay attack even in a single session! A counterexample is as follows:

$$b1(A).\overline{b2}\, N_B.b3(N_B).\overline{b4}\,(B, \{A, N_B\}_{\mathtt{k}[B,S]}).b5(\{A, N_B\}_{\mathtt{k}[B,S]}).\overline{acc}\, N_B$$

which can be interpreted as the following attack:

$$\begin{array}{ll} I(A) \longrightarrow B: & A \\ B \longrightarrow I(A): & N_B \\ I(A) \longrightarrow B: & N_B \\ B \longrightarrow I(S): & B, \{A, N_B\}_{K_{BS}} \\ I(S) \longrightarrow B: & \{A, N_B\}_{K_{BS}} \end{array}$$

One of the correct modifications of Woo-Lam protocol is to correct the message in the third flow, A sends an encrypted message whose plain message is not only N_B but also A's and B's names. Such a modification can prevent both two replay attacks we introduced above.

$$\begin{array}{ll} A \longrightarrow B: & A \\ B \longrightarrow A: & N_B \\ A \longrightarrow B: & \{A, B, N_B\}_{K_{AS}} \\ B \longrightarrow S: & B, \{A, \{A, B, N_B\}_{K_{AS}}\}_{K_{BS}} \\ S \longrightarrow B: & \{A, B, N_B\}_{K_{BS}} \end{array}$$

5.6 Compacting Parametric Traces with Type

Although the number of parametric traces generated by the parametric model for a security protocol in bounded sessions is finite, it is still too large to be handled, with lots of redundant parametric traces. For example, considering the following parametric transitions,

$$\begin{array}{l} \langle \epsilon, a(x).case\ x\ of\ \{y\}_{\mathtt{k}[A,B]}\ in\ P' \rangle \longrightarrow_p \\ \quad \langle a(x), case\ x\ of\ \{y\}_{\mathtt{k}[A,B]}\ in\ P' \rangle \longrightarrow_p \\ \quad\quad \langle a(\{y\}_{\mathtt{k}[A,B]}), P'\{\{y\}_{\mathtt{k}[A,B]}/x\} \rangle \end{array}$$

There are two parametric traces generated currently, $a(x)$ and $a(\{y\}_{\mathtt{k}[A,B]})$. Actually, $a(x)$ is a redundant parametric trace, since in reality, an input action and its following validating actions, such as decryption, splitting, matching, can be regarded an atomic action, among which no attacks can occur. In the above example, it is sufficient to only check whether $a(\{y\}_{\mathtt{k}[A,B]})$ satisfies a given action term.

To reduce the redundant parametric traces, a statical analysis on a process that describes a security protocol is proposed, gathering the information of each input variable by a *type system* [96, 97], then translating the process to its corresponding *parametric process* according to the type. By a simple semantics, the parametric process generates parametric traces. The set of parametric traces generated by this semantics is a proper subset of parametric traces generated by the parametric model in Figure 5.3. We find a corresponding concrete model with type restriction, which is sound and complete with respective to the representation. In the above example, its corresponding parametric process will be

$$a(\{y\}_{\mathtt{k}[A,B]}).case\ \{y\}_{\mathtt{k}[A,B]}\ of\ \{y\}_{\mathtt{k}[A,B]}\ in\ P'\{\{y\}_{\mathtt{k}[A,B]}/x\}$$

which only generates one parametric trace, $a(\{y\}_{\mathtt{k}[A,B]})$.

For details of this compaction approach, please refer to Appendix C.

Chapter 6

Authentication in Recursive Protocols

Generally, checking whether a security protocol in unbounded concurrent sessions satisfies a security property, such as secrecy, is undecidable [51, 50].

This chapter makes the first step toward model checking of security protocols with unbounded number of sessions. It allows to analyze protocols with one recursive procedure, which can naturally represents recursive protocols.

We adopt the full primitives defined in Definition 2.2. However, the following two restrictions are enforced to identifiers when defining a system.

- A system is restricted to only contain one *recursive process*;
- The expression that defines the recursive process is *sequential*.

We adopt the *pushdown system* (referred to as PDS) [105], an infinite state system with a finite set of control locations and an unbounded stack memory that can perform model checking on regular properties, to represent unbounded number of parametric traces.

The PDS cannot check all security properties represented by action terms in Subsection 5.2. Thus, a subset of action terms is defined to describe authentication property for the recursive protocols. These action terms can be checked in the pushdown system, which also enjoys the soundness and completeness.

Note that even under the assumption, secrecy property is also undecidable [75, 111]. [75, 111] analyze the secrecy property with bounded number of principals (thus bounded number of sessions), which can be straightforwardly analyzed by our method provided in Chapter 5.

6.1 Sub-Calculus for Recursive Protocols

To describe recursive protocols, we adopt the full primitives defined in Chapter 2.

Definition 6.1 (Processes for recursive protocols)**.** Let $\mathcal{P}$ be a countable set of processes

which is indicated by $P, Q, R, \ldots$. The syntax of processes is defined as follows:

$P, Q, R ::=$	
$\mathbf{0}$	Nil
$\overline{a}M.P$	output
$a(x).P$	input
$[M = N]\, P$	match
$(\texttt{new}\ x : \mathcal{A})P$	new
$(\nu\, n)P$	restriction
$let\ (x, y) = M\ in\ P$	pair splitting
$case\ M\ of\ \{x\}_L\ in\ P$	decryption
$P \| Q$	composition
$P + Q$	summation
$P; Q$	sequence
$\mathbb{A}(\tilde{pr})$	identifier

However, the following restrictions will be performed when defining a recursive process in order to check properties in the pushdown system.

- A system is restricted to only contain one recursive process (see Definition 2.5);
- The expression that defines the recursive process is sequential (see Definition 2.6).

For instance, SYS^{RA} defined in Subsection 3.4.2 is this kind of system, in which $\mathbb{O}(\texttt{A[Null]}, \texttt{A[A[Null]]})$ and S are flat processes, and $\mathbb{R}(\texttt{A[A[Null]]}, \texttt{A[Null]})$ is a recursive process.

Remark 6.1. The essence of the above restriction is, there only permits one recursive process in a system, composed with finite flat processes. In other words, it allows some finite processes run concurrently with a sequential recursive process. The system under this restriction is obvious weaker than a replication, $!P$. But it is more expressive than systems for analyzing recursive protocols proposed in [94, 34].

With the restrictions, the corresponding parametric model of a concrete model can be described by a pushdown system.

6.2 Representing Authentication for Recursive Protocols

6.2.1 A Subset of Action Terms

We we only use a subset of terms to define security properties for the RA protocol, defined as follows.

Definition 6.2. Let α and β be actions, with $f_v(\alpha) \subseteq f_v(\beta)$, and let s be a trace. We use $s \models \alpha \hookleftarrow \beta$ to represent that for each ground substitution ρ, if $\beta\rho$ occurs in s, then there exists one $\alpha\rho$ in s before $\beta\rho$. A configuration satisfies $\alpha \hookleftarrow \beta$, denoted by $\langle s, P\rangle \models \alpha \hookleftarrow \beta$, if each trace s' generated from $\langle s, P\rangle$ satisfies $s' \models \alpha \hookleftarrow \beta$.

Remark 6.2. Not all action terms can be checked in the pushdown system, especially negation of an action, and $\hookrightarrow_F$ between two actions.

- Given a concrete configuration $\mathcal{C}$, and a negation of an action, $\neg\alpha$, $\mathcal{C} \models \neg\alpha$ intuitively means that for any ground substitution ρ, $\alpha\rho$ does not occur in any trace s generated by $\mathcal{C}$. If α shares the same label with an action in the recursive process, then this action may repeatedly occur in generated traces unbounded number of times. This leads $\mathcal{C} \models \neg\alpha$ to be undecidable, which is the reason that the secrecy property for recursive protocols is undecidable [75, 111] in unbounded number of sessions.

- $\hookrightarrow_F$ is defined for a liveness problem, which solved by a reachability problem in a model without identifiers, since in this model, all transitions will terminate lastly. It is enough to only check the property on each terminated trace (see Lemma 5.2). However, when we try to check $\hookrightarrow_F$ in a non-terminate model, it fails to be solved by the similar approach introduced in Chapter 5. Current, whether $\hookrightarrow_F$ can be checked in a pushdown system is still unclear (It may or may not be solved by the LTL model checking on pushdown system, say, P-automaton in [105]). Here we tentatively avoid such a definition.

A definition in a parametric model that simulates the definition above in a is given as follows. By the Theorem 6.1 we know that an action term defined in a concrete model can be checked in a parametric model.

Definition 6.3. Let α and β be parametric actions, with $f_v(\alpha) \subseteq f_v(\beta)$, and let $\hat{s}$ be a parametric trace that has concretizations. $\hat{s} \models \alpha \hookleftarrow \beta$, if $s \models \alpha \hookleftarrow \beta$ for each concretization s of $\hat{s}$. We say that a parametric configuration satisfies $\alpha \hookleftarrow \beta$, denoted by $\langle \hat{s}, P\rangle \models \alpha \hookleftarrow \beta$, if $\hat{s}' \models \alpha \hookleftarrow \beta$ for each trace $\hat{s}'$ generated by $\langle \hat{s}, P\rangle$.

Theorem 6.1. *Given a parametric trace $\hat{s}$, $\hat{s} \models \alpha \hookleftarrow \beta$ if and only if, α is $\hat{\rho}$-unifiable in $\hat{s}$, and for each satisfiable normal form in $\mathtt{snf}_{\rightsquigarrow}(\hat{s})$ satisfying $\hat{s}\hat{\rho}'$, if $\beta\hat{\rho}'\hat{\rho}$ occurs in $\hat{s}\hat{\rho}'$ for some $\hat{\rho}$, then $\alpha\hat{\rho}'\hat{\rho}$ occurs before $\beta\hat{\rho}'\hat{\rho}$ in $\hat{s}\hat{\rho}'$.*

It is a corollary of the Theorem 5.3.

6.2.2 Representing Security Properties for the RA Protocol

To define authentication for the RA protocol (see Subsection 3.4.2), we need to insert two processes, $\overline{acc}\, M_1.\mathbf{0}$, and $\overline{acc}_r\, M_2.\mathbf{0}$ to the description of the RA protocol, which describe that the originator and the recipients declare they have received messages from their respective previous principals after they validate the messages. For the RA protocol, M_1 is the message received by $a2$, and M_2 is the message received by $b4$. A process for the RA protocol SYS_a^{RA} is transformed as follows, in which S is the original description represented in Subsection 3.3.1.

$$\begin{aligned}
\mathbb{O}_a(x_1,x_2) \triangleq & \overline{a1}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1,x_2,\mathtt{N}[\mathtt{Null}],\mathtt{Null}).a2(x).case\ x\ of\ \{y_1,y_2,y_3\}_{\mathtt{lk}[x_1,S]}.\\
& [y_3=\mathtt{N}[\mathtt{Null}]]\,\overline{\mathbf{acc}}\,\mathbf{x}.\mathbf{0}\\
\mathbb{R}_a(x_1,x_2) \triangleq & (b1(x).let\ (y_1,y_2,y_3,y_4,y_5)=x\ in\ [y_2=x_1]\\
& \overline{b2}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1,\mathtt{A}[x_1],\mathtt{N}[y_3],x).(\mathbb{R}(\mathtt{A}[x_1],x_1)+\overline{b3}\,\mathcal{H}_{\mathtt{lk}[x_1,S]}(x_1,S,\mathtt{N}[y_3],x).\mathbf{0}));\\
& (b4(x).let\ (z_1,z_2,z_3)=x\ in\\
& case\ z_1\ of\ \{z_4,z_5,z_6\}_{\mathtt{lk}[x_1,S]}\ in\ [z_5=\mathtt{A}[x_1]]\,[z_6=\mathtt{N}[y_3]]\\
& case\ z_2\ of\ \{z_7,z_8,z_9\}_{\mathtt{lk}[x_1,S]}\ in\ [z_8=x_2]\,[z_9=\mathtt{N}[y_3]]\,\overline{b5}z_3.\overline{\mathbf{acc}}_{\mathbf{r}}\,\mathbf{x}.\mathbf{0})\\
SYS_a^{RA} \triangleq & \mathbb{O}_a(\mathtt{A}[\mathtt{Null}],\mathtt{A}[\mathtt{A}[\mathtt{Null}]])\|\mathbb{R}_a(\mathtt{A}[\mathtt{A}[\mathtt{Null}]],\mathtt{A}[\mathtt{Null}])\|S
\end{aligned}$$

The authentication property of the RA protocol between the originator and its recipient is characterized formally as follows. Similarly, authentication between two adjacent recipients is also characterized formally.

Characterization 6.1. *[Authentication for the RA protocol I] Given the formal description of the RA protocol, the recipient is correctly authenticated to the originator, if*

$$\langle \epsilon, SYS\rangle \models \overline{b5}\,x \hookleftarrow \overline{acc}\,x$$

Characterization 6.2 (Authentication for the RA protocol II)**.** *Given the formal description of the RA protocol, the recipient is correctly authenticated to its previous recipient, if*

$$\langle \epsilon, SYS\rangle \models \overline{b5}\,x \hookleftarrow \overline{acc}_r\,x$$

6.3 Model Checking by the Pushdown System

By introducing the pushdown system, unbounded number of parametric traces in a parametric model can be represented by a finite set of control locations, with an unbounded stack memory.

Definition 6.4 (Pushdown system [105])**.** A pushdown system $\mathcal{P}=(Q,\Gamma,\Delta,c_0)$ is a quadruple, where Q contains the control locations, and Γ is the stack alphabet. A configuration of $\mathcal{P}$ is a pair (q,ω) where $q\in Q$ and $\omega\in\Gamma^*$. The set of all configurations is denoted by $\mathtt{conf}(\mathcal{P})$. With $\mathcal{P}$ we associated the unique transition system $\mathcal{I}_{\mathcal{P}}=(\mathtt{conf}(\mathcal{P}),\Rightarrow,c_0)$, whose initial configuration is c_0.

Δ is a finite subset of $(Q\times\Gamma)\times(Q\times\Gamma^*)$. If $((q,\gamma),(q',\omega))\in\Delta$, we also write $\langle q,\gamma\rangle\hookrightarrow\langle q',\omega\rangle$. For each transition relation, if $\langle q,\gamma\rangle\hookrightarrow\langle q',\omega\rangle$, then $\langle q,\gamma\omega'\rangle\Rightarrow\langle q',\omega\omega'\rangle$ for all $\omega'\in\Gamma^*$.

We define a set of messages used for the pushdown system as follows,

Definition 6.5 (Messages in the pushdown system)**.**

$$\begin{aligned}
pr ::= &\ n \mid \top \mid x \mid \mathtt{m}[\,] \mid \mathtt{m}[pr,\ldots,pr]\\
M,N,L ::= &\ pr \mid (M,N) \mid \{M\}_L \mid \mathcal{H}(M)
\end{aligned}$$

Two new messages are introduced in the pushdown system: $\top$ is a special name that denotes any name. It is used to substitute a variable which can be substituted to unbounded number of names during the refinement step. $\mathtt{m}[\,]$ is a *binder marker*. In the pushdown system encoding, we use the stack depth and binder markers to represent unbounded fresh messages. For instance, $\mathtt{A}[\mathtt{A}[\mathtt{Null}]]$ is represented by $\mathtt{A}[\,]$ and two stack elements in the stack.

Definition 6.6 (compaction)**.** Given a parametric trace $\hat{s}$, a compaction $\hat{tr}$ of $\hat{s}$ is a parametric trace cutting off redundant actions with the same labels.

We represent the parametric model by the PDS as follows,

- control locations are pairs $(R, \hat{tr})$, where R is a finite set of recursive messages, and $\hat{tr}$ is a compaction.
- stack alphabet only contains a symbol $\star$ to describe the depth of the recursive process;
- initial configuration is $\langle(\emptyset, \epsilon), \varepsilon\rangle$, where ϵ represents an empty parametric trace, and ε represents an empty stack;
- Δ are defined by two sets of translations, the translations for the parametric rules, and the translations for the refinement step.

An occurrence of **0** in the last sequential process of a recursive process means the return point of the current process.

We will replace it to a distinguished marker, **Nil**, when encoding a parametric system to the PDS.

For the parametric transition rules in Fig. 4.1, the transition rules in Δ are defined as follows, in which $\hat{tr}$ and $\hat{tr}'$ are compactions of $\hat{s}$ and $\hat{s}'$, respectively.

1. For parametric transition rules except $TOUTPUT$ and $TIND$ rules, $\langle(R, \hat{tr}), \omega\rangle \hookrightarrow \langle(R, \hat{tr}'), \omega\rangle$ if $\langle\hat{s}, P\rangle \longrightarrow_p \langle\hat{s}', P'\rangle$;
2. For $TOUTPUT$ rule, $\langle(R, \hat{tr}), \omega\rangle \hookrightarrow \langle(R', \hat{tr}.\overline{a}\,M), \omega\rangle$ if $\langle\hat{s}, \overline{a}\,M.P\rangle \longrightarrow_p \langle\hat{s}.\overline{a}\,M, P\rangle$, where $R' = R \cup \{M\}$ if M is a recursive message, otherwise $R' = R$.
3. For $TIND$ rule, when $\mathbb{R}$ is firstly met, $\langle(R, \hat{tr}), \omega\rangle \hookrightarrow \langle(R, \hat{tr}'), \omega\rangle$ if $\langle\hat{s}, P\rangle \longrightarrow_p \langle\hat{s}', P'\rangle$, where $\mathbb{R}(\tilde{pr}) \triangleq P$; Otherwise $\langle(R, \hat{tr}), \omega\rangle \hookrightarrow \langle(R, \hat{tr}), \star\omega\rangle$.
4. $\langle(R, \hat{tr}), \gamma\rangle \hookrightarrow \langle(R, \hat{tr}), \varepsilon\rangle$ if $\langle\hat{s}, \mathbf{Nil}\rangle$ is met.

For the refinement step, we have to distinguish two kinds of rigid messages, *context-insensitive* rigid messages, and *context-sensitive* rigid messages. Let's first take the following two examples.

Example 6.1. Consider the following simple example,

$$A \longrightarrow B : \qquad \{A, N_A\}_{K_{AB}}$$

A sends B its name and a nonce N_A under the encryption of K_{AB}. We assume that the protocol run recursively. The requirement of B (represented by a rigid message $\{A, x\}_{\mathtt{k}[A,B]}$) is insensitive to the current context, since B does not know which session the message it received belongs to. Thus the rigid message $\{A, x\}_{\mathtt{k}[A,B]}$ is a context-insensitive message.

Example 6.2. Consider the following simple example,

$$\begin{aligned} A \longrightarrow B: &\quad N_A \\ B \longrightarrow A: &\quad \{B, N_A\}_{K_{AB}} \end{aligned}$$

A sends B a nonce N_A, then requires B sending back B's name and the nonce under the encryption of K_{AB}. We assume that the protocol run recursively. The requirement of A (represented by a rigid message $\{x, \mathtt{N_A}[\,]\}_{\mathtt{k}[A,B]}$) is sensitive to the current context, since it knows the information of N_A, and thus knows which session the message it received belongs to. Thus the rigid message $\{x, \mathtt{N_A}[\,]\}_{\mathtt{k}[A,B]}$ is a context-insensitive message.

Definition 6.7 (Context-sensitive and context-insensitive rigid messages)**.**
Literally, context-sensitive rigid messages are rigid messages that contain at least one binder marker, while context-insensitive rigid messages do not contain any binder markers.

In other words, context-sensitive rigid messages can only be unified to finitely many atomic messages in the current session, since in one session, the number of atomic messages is finite. Context-insensitive rigid messages can be unified to infinitely many atomic messages over different sessions.

The transition relations for the refinement step in Definition 4.5 in Δ are defined as follows. If $\hat{tr} = \hat{tr}_1.a(M).\hat{tr}_2$, such that there exist a rigid message N in M such that $N \notin el(\hat{tr}_1)$.

5. $\langle (R, \hat{tr}), \omega \rangle \hookrightarrow \langle (R, \hat{tr}\hat{\rho}), \omega \rangle$, if N is context-sensitive and $\hat{\rho}$-unifiable in $R \cup el(\hat{s}_1)$.

6. $\langle (R, \hat{tr}), \omega \rangle \hookrightarrow \langle (R \cup N', \hat{tr}\hat{\rho}'), \omega \rangle$, if N is context-insensitive and $\hat{\rho}$-unifiable to N' in $el(\hat{s}_1)$, and $\hat{\rho}'$ is the substitution that replaces different elements in N and N' with $\top$.

Lemma 6.2. *Given an initial configuration $\langle \epsilon, P \rangle$, for each parametric trace $\hat{s}$ generated by $\langle \epsilon, P \rangle$, each satisfiable normal form $\hat{s}\hat{\rho}$ of $\hat{s}$ has a compaction $\hat{tr}'$ in a control location of its corresponding PDS.*

Proof. By induction, it is easy to proof that each parametric trace generated by parametric transition rules in Figure 4.1 has a compaction.

According to Definition 4.5, satisfiable normal forms are generated by unifying rigid messages with atomic messages in the prefix parametric trace. Let's do case analysis on rigid messages.

1. Case current rigid message is a context-sensitive rigid message, then its can only be satisfied by atomic messages within the current session. The number of the atomic messages is finite. The transition relation 5 in the PDS is essentially unifying infinitely many context-sensitive rigid messages with the same pattern in one turn. It does not lose any probabilities for unification, since any atomic messages that can be unified in $\hat{s}'$ are explicitly represented in $\hat{tr}'$.

2. Case current rigid message is a context-insensitive rigid message, then it can be satisfied by infinitely many atomic message over different sessions. The number of these atomic messages is infinite. But with finite patterns. The difference of atomic messages with the same pattern is only the difference of fresh names in each message.

Firstly, if there exist infinitely many atomic messages that satisfies the rigid message, then there must exist an atomic message in the current session. So when a context-insensitive rigid message in $\hat{tr}'$ is met and can be unified to a message in its prefix parametric trace, we can expect it can be unified to infinitely many these messages over different sessions with the same pattern. According to transition relation 6 in the PDS, the place that will be substituted to any fresh message will be marked as a $\top$. It also does not lose any probabilities for unification.

Thus, each satisfiable normal form $\hat{s}\hat{\rho}$ of $\hat{s}$ generated by $\langle \epsilon, P \rangle$ has a compaction $\hat{tr}'$ in a control location of its corresponding PDS.

□

Theorem 6.3. *Given an initial configuration $\langle \epsilon, P \rangle$, and two actions α and β, which satisfies $f_v(\alpha) \subseteq f_v(\beta)$, $\langle \epsilon, P \rangle \models \alpha \hookleftarrow \beta$ if and only if, for each control location $(R, \hat{tr})$ of its corresponding PDS, $\hat{tr}$ is a satisfiable normal form, and $\hat{tr} \models \alpha \hookleftarrow \beta$.*

Proof. By induction, it is easy to proof that each parametric trace generated by parametric transition rules in Fig. 4.1 has a compaction. Furthermore, by Lemma 6.2, each satisfiable normal form $\hat{s}\hat{\rho}$ of $\hat{s}$ generated by $\langle \epsilon, P \rangle$ has a compaction $\hat{tr}'$ in a control location of its corresponding PDS.

According to Theorem 6.1, we know that $\hat{s} \models \alpha \hookleftarrow \beta$ if each satisfiable normal form $\hat{s}'$ satisfies, $\alpha\hat{\rho}'\hat{\rho}$ occurs before $\beta\hat{\rho}'\hat{\rho}$ in $\hat{s}\hat{\rho}'$ if $\beta\hat{\rho}'\hat{\rho}$ occurs in $\hat{s}\hat{\rho}'$ for some $\hat{\rho}$. So it is sufficient to prove that each satisfiable normal form $\hat{s}' \models \alpha \hookleftarrow \beta$ if and only if $\hat{tr}' \models \alpha \hookleftarrow \beta$. "$\Leftarrow$" is obvious, since $\hat{tr}'$ is a sub-trace of $\hat{s}'$ ($\top$ can be represented to the fresh names in the current session).

"$\Rightarrow$": If one satisfiable normal form $\hat{s}$ satisfies $\hat{s} \models \alpha \hookleftarrow \beta$, then there exists a $\hat{\rho}$ such that $\alpha\hat{\rho}$ occurs in $\hat{s}$ before $\beta\hat{\rho}$. Let's do case analysis on β.

There are only two ways to make two actions have common sub-messages, say, satisfies $\hat{s} \models \alpha \hookleftarrow \beta$.

1. The first one is due to sequence actions in a process. The process may receive a message by α, then send a new message with the common sub-messages to previous message by β. In this case, it is simple to show that $\hat{s} \models \alpha \hookleftarrow \beta$ implies $\hat{tr} \models \alpha \hookleftarrow \beta$, since all distinguished actions are contained in $\hat{tr}$.

2. The second one is due to refinement step, there exists a rigid message in β (it is not necessary that the rigid message firstly occurs in β), then by unification with previous atomic messages, the message in β has common sub-messages with α. If the rigid message is a context-sensitive one, then the α must occur in the current session, thus in $\hat{tr}$. If it is a context-insensitive one, although the α that shares common sub-messages does not occur in the current session, there at least exists one α satisfies the property.

Thus, $\langle \epsilon, P \rangle \models \alpha \hookleftarrow \beta$ can be checked in its corresponding PDS. □

6.4 Attacks of the RA Protocol and Its Modification

There are two works on analyzing authentication property for the RA protocol with bounded number of principals based on higher order theorem proving [94, 34]. One was

proposed by L. Paulson, using Isabelle/HOL [94]. The other was proposed by J. Bryans and S. Schneider, adopting CSP, and using PVS as a theorem prover.

Paulson took a different definition of authentication; Bryans and Schneider considered authentication between the server and the last principal who submitted the request. Their works proved that the RA protocol satisfies the authentication property with bounded number of principals.

We took the same definition of authentication in [**?**], which is different from Paulson's. An attack that violates our definition is detected by our formalism. In the next subsection, we will discuss different points of view of the the attack we detected.

For the recursive authentication protocol, counterexamples that violate authentication characterized in Characterization 6.1 are found in compactions of its parametric traces. One of them is shown as follows:

$$\overline{a1}\,\mathcal{H}_{\mathtt{lk[A[]},S]}(\mathtt{A[]},\mathtt{A[A[]]},\mathtt{N[]},\mathtt{Null}).b1(\mathcal{H}_{\mathtt{lk[A[]},S]}(\mathtt{A[]},\mathtt{A[A[]]},\mathtt{N[]},\mathtt{Null})).$$
$$\overline{b3}\,\mathcal{H}_{\mathtt{lk[A[A[]]},S]}(\mathtt{A[A[]]},S,\mathtt{N[N[]]},\mathcal{H}_{\mathtt{lk[A[]},S]}(\mathtt{A[]},\mathtt{A[A[]]},\mathtt{N[]},\mathtt{Null})).$$
$$s1\mathcal{H}_{\mathtt{A[A[]]},S]}(\mathtt{A[A[]]},S,\mathtt{N[N[]]},\mathcal{H}_{\mathtt{lk[A[]},S]}(\mathtt{A[]},\mathtt{A[A[]]},\mathtt{N[]},\mathtt{Null})).$$
$$\overline{s2}\,(\{\mathtt{k[k[]]},S,\mathtt{N[N[]]}\}_{\mathtt{lk[A[A[]]},S]},\{\mathtt{k[]},\mathtt{A[]},\mathtt{N[N[]]}\}_{\mathtt{lk[A[A[]]},S]},\{\mathtt{k[]},\mathtt{A[A[]]},\mathtt{N[]}\}_{\mathtt{lk[A[]},S]}).$$
$$a2(\{\mathtt{k[]},\mathtt{A[A[]]},\mathtt{N[]}\}_{\mathtt{lk[A[]},S]}).\overline{acc}\,\{\mathtt{k[]},\mathtt{A[A[]]},\mathtt{N[]}\}_{\mathtt{lk[A[]},S]}$$

This counterexample actually represents infinitely many attacks, due to the number of principals can be arbitrarily large. The minimal one is as follows,

$$\begin{array}{rcll} A_0 & \longrightarrow & A_1: & \mathcal{H}_{K_{A_0S}}(A_0,A_1,N_{A_0},\mathtt{Null}) \\ A_1 & \longrightarrow & S: & \mathcal{H}_{K_{A_1S}}(A_1,S,N_{A_1},\mathcal{H}_{K_{A_0S}}(A_0,A_1,N_{A_0},\mathtt{Null})) \\ S & \longrightarrow & I(A_1): & \{K_1,S,N_{A_1}\}_{K_{A_1S}},\{K_0,A_0,N_{A_1}\}_{K_{A_1S}},\{K_0,A_1,N_{A_0}\}_{K_{A_0S}} \\ I(A_1) & \longrightarrow & A_0: & \{K_0,A_1,N_0\}_{K_{A_0S}} \end{array}$$

The reason that attacks occur is that S directly sends all messages without any protections to the recipient who submits requirement. Thus an intruder can steal these messages, and pretend each recipient to authenticate to its previous recipient one by one, although the intruder cannot know contexts of messages. One possible modification is that S protects the message it sends by encrypting it iteratively with each symmetric key shared with each recipient. However, the refinement is quite time consuming, and thus is inefficient. We use three principals to illustrate the modification, which can be straightforwardly applied to the one with infinitely many but bounded number of principals.

$$
\begin{aligned}
A_0 &\longrightarrow A_1 : && \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null}) \\
A_1 &\longrightarrow A_2 : && \mathcal{H}_{K_{A_1S}}(A_1, A_2, N_{A_1}, \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null})) \\
A_2 &\longrightarrow S : && \mathcal{H}_{K_{A_2S}}(A_2, S, N_{A_2}, \mathcal{H}_{K_{A_1S}}(A_1, A_2, N_{A_1}, \mathcal{H}_{K_{A_0S}}(A_0, A_1, N_{A_0}, \texttt{Null}))) \\
S &\longrightarrow A_2 : && \{\{K_2, S, N_{A_2}\}_{K_{A_2S}}, \{K_1, A_1, N_{A_2}\}_{K_{A_2S}}, \\
& && \{\{K_1, A_2, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_0, N_{A_1}\}_{K_{A_1S}}, \\
& && \{K_0, A_1, N_{A_0}\}_{K_{A_0S}}\}_{K_{A_1S}}\}_{K_{A_2S}} \\
A_2 &\longrightarrow A_1 : && \{\{K_1, A_2, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_0, N_{A_1}\}_{K_{A_1S}}, \{K_0, A_1, N_{A_0}\}_{K_{A_0S}}\}_{K_{A_1S}} \\
A_1 &\longrightarrow A_0 : && \{K_0, A_1, N_0\}_{K_{A_0S}}
\end{aligned}
$$

6.5 Different Points of View to the Attack

Before we discuss different points of view to the attack we have detected, let us take an example.

Example 6.3. Let's consider the fixed NSPK protocol as our first example, which is showed as follows.

$$
\begin{aligned}
A &\longrightarrow B : && \{A, N_A\}_{+K_B} \\
B &\longrightarrow A : && \{B, N_A, N_B\}_{+K_A} \\
A &\longrightarrow B : && \{N_B\}_{+K_B}
\end{aligned}
$$

After numerous verification applied to the protocol, it can guarantee that the fixed NSPK protocols satisfies authentication property. However, intruder can has the following action: it intercepts the message from A, then sends the same message to B, as follows:

$$
\begin{aligned}
A &\longrightarrow B : && \{A, N_A\}_{+K_B} \\
B &\longrightarrow A : && \{B, N_A, N_B\}_{+K_A} \\
A &\longrightarrow I(B) : && \{N_B\}_{+K_B} \\
I(A) &\longrightarrow B : && \{N_B\}_{+K_B}
\end{aligned}
$$

It is certainly not an attack. Otherwise any protocol will suffer with such kind of "attacks". By our definition, it is not an attack, since when B received the message, and "thought" it comes from A. Then A really sent the same message. No matter how B got the message.

Now, comparing with the example above, let's consider the following example.

Example 6.4. The original Otway-Rees protocol is defined flow-by-flow as follows.

$$
\begin{aligned}
A &\longrightarrow B : && M, A, B, \{N_A, M, A, B\}_{K_{AS}} \\
B &\longrightarrow S : && M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}} \\
S &\longrightarrow B : && M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
B &\longrightarrow A : && M, \{N_A, K_{AB}\}_{K_{AS}}
\end{aligned}
$$

This protocol has an attack [36], an attacker can intercept the message sent by S, split it, and sends it to A. Hence when A gets the message, and "thinks" it comes from B, but B never sent the message, since it did not accept the message from S. The attack is described as follows.

$$\begin{aligned} A \longrightarrow B: &\quad M, A, B, \{N_A, M, A, B\}_{K_{AS}} \\ B \longrightarrow S: &\quad M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}} \\ S \longrightarrow I(B): &\quad M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\ I(B) \longrightarrow A: &\quad M, \{N_A, K_{AB}\}_{K_{AS}} \end{aligned}$$

The attack for the RA protocol we detected is such kind of attack. It violates our definition for the authentication.

Abadi et. al. thought this attack really caused loss for principals: "It is interesting to note that this protocol does not make use of K_{AB} as an encryption key, so neither principal can know whether the key is known to the other [36]."

However, Paulson, et. al. did not agree on the above point: "We refute the claim, showing that there exists a protocol similar to Otway-Rees that does not use the session key as an encryption key but informs one agent that his peer does know the session key [18]."

In Paulson's view, the attack in Example 6.4 causes the same result as the attack in Example 6.3. Actually, even in the fixed NSPK protocol, under the assumption of network with intruders, A cannot guarantee that B eventually get the last message, so that they cannot update the N_B as a session key for later communications.

J. Bryans and S. Schneider took the similar definition of authentication as ours. They proved the authentication between the server and the last principal.

Chapter 7

Non-repudiation and Fairness in Bounded Sessions

Fair non-repudiation protocols intend reliable exchange of messages in the situation that each principal can be dishonest, by trying to take advantages from other principals by aborting the communication or sending fake messages. A fair non-repudiation protocol needs to ensure two properties, non-repudiation and fairness. Non-repudiation means that when a sender sends a message to a receiver, neither the sender nor the receiver can deny the participation in this communication. Fairness means no principals can obtain evidence while the other principal cannot do so. Finding flaws for these properties is more difficult than for secrecy and authentication due to misbehaviors of dishonest principals.

This chapter extends the concrete model in Chapter 5 that describes authentication protocols in bounded sessions, with a deductive system to generate infinitely many messages that dishonest principals may produce and send [103]. Furthermore, summations will be adopted to describe different choice of a dishonest principal.

A finite parametric model is proposed by abstracting all infinities, following the similar approach in Chapter 5. The finite parametric traces generated by the parametric model has the same representative ability as its corresponding concrete model. Thus other security properties, such as non-repudiation and fairness, can be checked in the parametric model.

7.1 Extended Model for Non-repudiation Protocols

7.1.1 Extended Process Calculus and Concrete Trace

To describe fair non-repudiation protocols, we use the same definition of messages that defined in Definition 2.1. For processes, the *new* primitive has modified, in which the range of the bound variable will decided dynamically through transitions. Furthermore, summation is adopted.

Definition 7.1 (Processes for fair non-repudiation protocols)**.** Let $\mathcal{P}$ be a countable set of

processes which is indicated by $P, Q, R, \ldots$. The syntax of processes is defined as follows:

$P, Q, R ::=$	
$\mathbf{0}$	Nil
$\overline{a}M.P$	output
$a(x).P$	input
$[M = N]\, P$	match
$(\texttt{new}\ x)P$	new
$(\nu\, n)P$	restriction
$let\ (x, y) = M\ in\ P$	pair splitting
$case\ M\ of\ \{x\}_L\ in\ P$	decryption
$P \| Q$	composition
$P + Q$	summation

The range of the variable x in the process $(\texttt{new}\ x)P$ is not decided statically. Instead, it will increase during transitions, which will be deduced dynamically by a deductive system. By the primitive *new*, we can describe all messages a dishonest principal can generate from the current finite knowledge.

The process, summation $P+Q$ is used here, since we need to describe that a dishonest principal in a fair non-repudiation protocol usually has several choices, such as aborting communication, or running a recovery stage, which makes a branching run possible.

A process P that describes a dishonest principal A can send out all messages generated through the environmental deductive system, $\vdash$ (see Figure 2.1), and can also encrypt messages with A's private key and shared key, and apply one-way hash function to the message. A *P-deductive system* is defined in Figure 7.1.

$$\frac{S \vdash M}{S \vdash_P M} \qquad \frac{S \vdash_P M}{S \vdash_P \{M\}_{\mathtt{k}[A,B]}}$$

$$\frac{S \vdash_P M}{S \vdash_P \{M\}_{-\mathtt{k}[A]}} \qquad \frac{S \vdash_P M}{S \vdash_P \mathcal{H}(M)}$$

Figure 7.1: P-deductive system

In this chapter, we will modify the definition of the *concrete trace* as following, which also restricts the message in output actions (comparing to Definition 2.8). The messages in a concrete trace s, represented by $msg(s)$, are those messages in output actions of the concrete trace s. We use $s \vdash M$ to abbreviate $msg(s) \vdash M$, and $s \vdash_P M$ to abbreviate $msg(s) \vdash_P M$.

Definition 7.2 (Concrete trace and configuration)**.** A concrete trace s is a ground action string that satisfies each decomposition $s = s'.a(M).s''$ implies $s' \vdash M$, and each $s = s'.\overline{a}M.s''$ implies $s' \vdash_P M$, where P is a closed process that contains the label a. ϵ represents an empty trace. A concrete configuration is a pair $\langle s, P\rangle$, in which s is a concrete trace and P is a closed process.

7.1.2 Operational Semantics

The transition relation of the extended concrete model is defined by the rules in Figure 7.2.

$$
\begin{array}{rl}
(INPUT) & \langle s, a(x).P\rangle \longrightarrow^e \langle s.a(M), P\{M/x\}\rangle \;\; s \vdash M \\
(OUTPUT) & \langle s, \overline{a}M.P\rangle \longrightarrow^e \langle s.\overline{a}M, P\rangle \\
(DEC) & \langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow^e \langle s, P\{M/x\}\rangle \;\; L' = \mathtt{Opp}(L) \\
(PAIR) & \langle s, let\ (x,y) = (M,N)\ in\ P\rangle \longrightarrow^e \langle s, P\{M/x, N/y\}\rangle \\
(NEW) & \langle s, (\mathtt{new}\ x)P\rangle \longrightarrow^e \langle s, P\{M/x\}\rangle \quad s \vdash_P M \\
(RESTRICTION) & \langle s, (\nu n)P\rangle \longrightarrow^e \langle s, P\{m/n\}\rangle \quad m = \mathtt{freshN}(V) \\
(MATCH) & \langle s, [M = M]P\rangle \longrightarrow^e \langle s, P\rangle \\
(LCOM) & \dfrac{\langle s, P\rangle \longrightarrow^e \langle s', P'\rangle}{\langle s, P\|Q\rangle \longrightarrow^e \langle s', P'\|Q\rangle} \\
(RCOM) & \dfrac{\langle s, Q\rangle \longrightarrow^e \langle s', Q'\rangle}{\langle s, P\|Q\rangle \longrightarrow^e \langle s', P\|Q'\rangle} \\
(LSUM) & \langle s, P+Q\rangle \longrightarrow^e \langle s, P\rangle \\
(RSUM) & \langle s, P+Q\rangle \longrightarrow^e \langle s, Q\rangle \\
(STR) & \dfrac{P \equiv P' \quad \langle s, P'\rangle \longrightarrow^e \langle s', Q'\rangle \quad Q' \equiv Q}{\langle s, P\rangle \longrightarrow^e \langle s', Q\rangle}
\end{array}
$$

Figure 7.2: Concrete Transition Rules for the Extended Model

By the rule NEW, a variable bound by `new` can be instantiated to infinitely many ground messages generated by a P-deductive system. Thus an output action with fresh bound variables may send any of these messages to the environment. This is a new factor that leads infinity of a system.

7.1.3 Describing Fair Non-repudiation Protocols

When model checking authentication or secrecy properties, we make assumptions that the legitimate principals are honest, i.e., behave following the prescription of a security protocol (see Chapter 2). For example, principals A and B want to have a private conversation, it is in their interests not to purposely disclose their keys and confidential messages.

Other security goals, such as non-repudiation and fairness, are rather different. We are concerned with protecting one principal against possible cheating by another. For example, a non-repudiation goal of transmission should provide the receiver with proof that the message was indeed sent by the claimed sender, even if the sender subsequently tries to deny it. Thus we cannot assume that the principal will not cheat.

To describe a case that a dishonest principal tries to cheat another principal by sending a fake message, fresh variables are used to denote the sub-message that the principal can use to deceive another principal. These variables are bound by the `new` primitive, and are later instantiated by some ground messages deduced by the P-deductive system according to the NEW transition rules.

We use a simplified variation of the *Zhou-Gollmann fair non-repudiation protocol* (referred to as the simplified ZG protocol) as a running example to illustrate how our system works. The full ZG protocol is described in [116]. Note that besides a standard flow description, a fair non-repudiation protocol also contains a description on what are evidences for participated principals.

In this protocol, A aims to send a message M to B. At the same time, it expects to obtain an evidence that the message was received by B, and B has an evidence that the message was sent by A. The message is transferred in two stages: an encrypted message protected by a new generated key K is first sent directly to B. After A has received evidence of receipt from B, the key K is sent via a trust third party S and both A and B can receive the evidence that K has been distributed by S. The simplified ZG protocol is described flow-by-flow as follows:

$$A \longrightarrow B: \quad \{B, N_A, \{M\}_K\}_{-K_A} \tag{7.1}$$
$$B \longrightarrow A: \quad \{A, N_A, \{M\}_K\}_{-K_B} \tag{7.2}$$
$$A \longrightarrow S: \quad \{B, N_A, K\}_{-K_A} \tag{7.3}$$
$$S \longrightarrow A: \quad \{A, B, N_A, K\}_{-K_S} \tag{7.4}$$
$$S \longrightarrow B: \quad \{A, B, N_A, K\}_{-K_S} \tag{7.5}$$

The evidence that A *sends the message* M *to* B (referred as M_1) is the pair of messages that B accepted in (6.1) and (6.5). In (6.1), A sends a signed message to B, and B can confirm that the intended receiver of (6.1) is B by decrypting it by the public key $+K_A$. In (6.5), B checks whether N_A in (6.5) coincides with that in (6.1). If they match, B can confirm that the TTP S has received K from A in (6.3). Alternatively, the evidence that B *receives the message* M *from* A (referred as M_2) is the pair of the messages that A accepted in (6.2) and (6.4).

Principal A will be represented in our calculus as follows:

$$\begin{aligned} A \triangleq & (\nu N_A)(\mathtt{new}\, x_1, x_2)\, \overline{a1}\{x_1, N_A, x_2\}_{-\mathtt{k}[A]}.a2(x_3). \\ & \mathit{case}\ x_3\ \mathit{of}\ \{x_4, x_5, x_6\}_{-\mathtt{k}[x_1]}\ \mathit{in}\ [x_4 = A]\,[x_5 = N_A]\,[x_6 = x_2]\,(\mathbf{0} + \\ & (\mathtt{new}\, x_7, x_8)\, \overline{a3}\{x_7, x_8\}_{-\mathtt{k}[A]}.a4(x_9).\ \mathit{case}\ x_9\ \mathit{of}\ \{x_{10}, x_{11}, x_{12}, x_{13}\}_{-\mathtt{k}[S]} \\ & \mathit{in}\ [x_{10} = A]\,[x_{11} = x_1]\,[x_{12} = N_A]\,[x_{13} = x_8].\mathbf{0}) \end{aligned}$$

A cannot have the information of who it will communicate with. Thus a binder is used to denote any possible principal it communicates with, in which x_1 is bound by $(\mathtt{new}\ x_1)$. Furthermore, in the first flow, A need not send the right encrypted message, following the prescription of the protocol. On the contrary, it can send any messages, denoted by a fresh bound variable x_2. After receiving a message by a_2, A may abort the communication, or send a message. When sending a message, similarly, it still does not follow the prescription of the protocol, sending two arbitrary messages denoted by two bound variables x_7 and x_8. The latter is used to check the validation of message he has received by $a4$.

$$\begin{aligned} B \triangleq & b1(y_1).\mathit{case}\ y_1\ \mathit{of}\ \{y_2, y_3, y_4\}_{-\mathtt{k}[A]}\ \mathit{in}[y_2 = B]\,(\mathbf{0} + \\ & (\mathtt{new}\, y_5)\, \overline{b2}\{A, y_5\}_{-\mathtt{k}[B]}.b3(y_6).\mathit{case}\ y_6\ \mathit{of}\ \{y_7, y_8, y_9, y_{10}\}_{-\mathtt{k}[S]}\ \mathit{in} \\ & [y_7 = A]\,[y_8 = B]\,[y_9 = y_3].\mathbf{0}) \end{aligned}$$

As a receiver, we also fix B's potential sender to the sender A in one session. Such an assumption is necessary when defining security properties, since otherwise the sender and the receiver we represented may have no connections with each other, and thus these properties cannot be defined between them. Similarly, after receiving a message, B may also quit the communication, or send a message, disobeying the prescription of the protocol. Thus a fresh variable y_5 is used, bound by a `new` operator.

$$S \triangleq s1(z_1).case\ z_1\ of\ \{z_2\}_{-\mathtt{k}[z_3]}\ in\ \overline{s2}\{z_3, z_2\}_{-\mathtt{k}[S]}.\overline{s2}\{z_3, z_2\}_{-\mathtt{k}[S]}.\mathbf{0}$$

TTP S is faithfully following the prescription of the protocol. Furthermore, the simplified ZG protocol can be described as a composition among A, B and S.

$$SYS^{ZG} \triangleq A \| B \| S$$

7.2 Representing non-repudiation and fairness

Action terms defined in Subsection 5.2 is also adopted to represent non-repudiation and fairness properties.

7.2.1 Non-repudiation

The non-repudiation property is that neither the sender nor the receiver can deny this after participating in this communication, when a sender sends some message to a receiver. Usually, it concerns the following two properties [115, 73]:

- *Non-repudiation of origin (NRO)* is intended to protect against the sender's false denial of having sent the messages.
- *Non-repudiation of receipt (NRR)* is intended to protect against the receiver's false denial of having received the message.

For non-repudiation and fairness, two processes, $\overline{evid}_B\, M_1.\mathbf{0}$ and $\overline{evid}_A\, M_2.\mathbf{0}$, are introduced for A and B, respectively. For the simplified ZG protocol, the evidence M_1 (resp. M_2) is the pair of messages in (1) and (5) (resp. (2) and (4)). In the protocol description, they are messages received at $b1$ and $b3$ (resp. $a2$ and $a4$) as y_1 and y_6 (resp. x_3 and x_9). Then the submitted process is $\overline{evid}_A(y_1, y_6).\mathbf{0}$ (resp. $\overline{evid}_B(x_3, x_9).\mathbf{0}$).

$$\begin{aligned}
A_p \triangleq & (\nu N_A)(\mathtt{new}\, x_1, x_2)\, \overline{a1}\{x_1, N_A, x_2\}_{-\mathtt{k}[A]}).a2(x_3).\\
& case\ x_3\ of\ \{x_4, x_5, x_6\}_{-\mathtt{k}[x_1]}\ in\ [x_4 = A]\,[x_5 = N_A]\,[x_6 = x_2]\,(\mathbf{0}+\\
& (\mathtt{new}\, x_7, x_8)\, \overline{a3}\{x_7, x_8\}_{-\mathtt{k}[A]}.a4(x_9).\ case\ x_9\ of\ \{x_{10}, x_{11}, x_{12}, x_{13}\}_{-\mathtt{k}[S]}\\
& in\ [x_{10} = A]\,[x_{11} = x_1]\,[x_{12} = N_A]\,[x_{13} = x_8].\overline{\mathbf{evid}}_{\mathbf{B}}(\mathbf{x_3}, \mathbf{x_9}).\mathbf{0})\\
B_p \triangleq & b1(y_1).case\ y_1\ of\ \{y_2, y_3, y_4\}_{-\mathtt{k}[A]}\ in [y_2 = B]\,(\mathbf{0}+\\
& (\mathtt{new}\, y_5)\, \overline{b2}\{A, y_5\}_{-\mathtt{k}[B]}.b3(y_6).case\ y_6\ of\ \{y_7, y_8, y_9, y_{10}\}_{-\mathtt{k}[S]}\ in\\
& [y_7 = A]\,[y_8 = B]\,[y_9 = y_3].\overline{\mathbf{evid}}_{\mathbf{A}}(\mathbf{y_1}, \mathbf{y_6}).\mathbf{0})\\
SYS_p^{ZG} \triangleq & A_p \| B_p \| S
\end{aligned}$$

For NRO, we guarantee that a receiver got the evidence of a sender should come after the sender sent the message. Thus $\hookleftarrow$ is used to describe the property. For NRR, we guarantee that after a sender got the evidence, his receiver will inevitably obtain the message. We will use $\hookrightarrow_F$ to characterize the property.

Characterization 7.1 (NRO for the simplified ZG protocol)**.** *Given the process of the simplified ZG protocol, the NRO is satisfied, if*

$$\langle \epsilon, SYS_p^{ZG} \rangle \models \overline{a1}\{B,x,y\}_{-\mathtt{k}[A]} \wedge \overline{a3}\{B,x,z\}_{-\mathtt{k}[A]} \hookleftarrow \overline{evid}_A(\{B,x,y\}_{-\mathtt{k}[A]}, \{A,B,x,z\}_{-\mathtt{k}[S]})$$

Characterization 7.2 (NRR for the simplified ZG protocol)**.** *Given the process of the simplified ZG protocol, the NRR is satisfied if*

$$\langle \epsilon, SYS_p^{ZG} \rangle \models \overline{evid}_B(\{A,x,y\}_{-\mathtt{k}[B]}, \{A,B,x,z\}_{-\mathtt{k}[S]}) \hookrightarrow_F \overline{b2}\{A,x,y\}_{-\mathtt{k}[B]} \wedge \overline{s2}\{A,B,x,z\}_{-\mathtt{k}[S]}$$

7.2.2 Fairness

A protocol is unfair, if one principal can obtain the evidence while the other principal cannot do so. Furthermore, a receiver cannot access the message until the sender has the evidence of the receiver. To define the fairness that satisfies the above requirements, we need the following three properties [103, 73].

- Fairness for origin obtaining evidence: If a receiver has an evidence of its sender, then the sender should have an evidence of the receiver. We name it FAIRO.
- Fairness for receipt obtaining evidence: If a sender has an evidence of its receiver, then the receiver should obtain an evidence of the sender. We name it FAIRR.
- Fairness for message receipt: A receiver should not know the message until the evidence of the receiver has been provided to the sender. We name it FAIRM.

Characterization 7.3 (FAIRO for the simplified ZG protocol)**.** *Given the process of the simplified ZG protocol, the FAIRO is satisfied if*

$$\langle \epsilon, SYS_p^{ZG} \rangle \models \overline{evid}_A(\{B,x,y\}_{-\mathtt{k}[A]}, \{A,B,x,z\}_{-\mathtt{k}[S]}) \hookrightarrow_F \overline{evid}_B(\{A,x,y\}_{-\mathtt{k}[B]}, \{A,B,x,z\}_{-\mathtt{k}[S]})$$

Characterization 7.4 (FAIRR for the simplified ZG protocol)**.** *Given the process of the simplified ZG protocol, the FAIRR is satisfied, if*

$$\langle \epsilon, SYS_p^{ZG} \rangle \models \overline{evid}_B(\{A,x,y\}_{-\mathtt{k}[B]}, \{A,B,x,z\}_{-\mathtt{k}[S]}) \hookrightarrow_F \overline{evid}_A(\{B,x,y\}_{-\mathtt{k}[A]}, \{A,B,x,z\}_{-\mathtt{k}[S]})$$

For FAIRM, a listener process, $check(x).\mathbf{0}$, is inserted in B_p, similar to when we define secrecy property. It is used to check whether the message is already leaked in the environment.

$$B_f \triangleq B_p \| \mathbf{check}(\mathbf{x}).\mathbf{0}$$
$$SYS_f^{ZG} \triangleq A_p \| B_c \| S$$

Characterization 7.5 (FAIRM for the simplified ZG protocol)**.** *Given the process of the simplified ZG protocol, the FAIRM is satisfied if*

$$\langle \epsilon, SYS_f^{ZG} \rangle \models \neg \overline{check}\, z \leftrightarrow \overline{evid}_B(\{A, x, y\}_{-\mathtt{k}[B]}, \{A, B, x, z\}_{-\mathtt{k}[S]})$$

7.3 Parametrization and Refinement

After extending the concrete model, we also similarly find a corresponding parametric model. It has the same representative ability as the extended concrete model. However, when refining the parametric trace a parametric model generate, not only messages in each input action, but also messages in each output action need to be considered.

7.3.1 Parametric Trace and Operational Semantics

The parametric model has the exactly same definition of parametric traces and configurations defined in Chapter 3. (see Definition 4.1).

Definition 7.3 (Parametric trace and configuration)**.** A *parametric trace* $\hat{s}$ is a string of actions. A *parametric configuration* is a pair $\langle \hat{s}, P \rangle$, in which $\hat{s}$ is a parametric trace and P is a process.

The transition relation is given by the rules in Figure 7.3.

$$
\begin{array}{rl}
(PINPUT) & \langle \hat{s}, a(x).P \rangle \longrightarrow_p^e \langle \hat{s}.a(x), P \rangle \\
(POUTPUT) & \langle \hat{s}, \overline{a}M.P \rangle \longrightarrow_p^e \langle \hat{s}.\overline{a}M, P \rangle \\
(PDEC) & \langle \hat{s}, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P \rangle \longrightarrow_p^e \langle \hat{s}\theta, P\theta \rangle \\
 & \qquad \theta = \mathtt{Mgu}(\{M\}_L, \{x\}_{\mathtt{Dpp}(L')}) \\
(PPAIR) & \langle \hat{s}, let\ (x,y) = M\ in\ P \rangle \longrightarrow_p^e \langle \hat{s}\theta, P\theta \rangle \quad \theta = \mathtt{Mgu}((x,y), M) \\
(PNEW) & \langle \hat{s}, (\mathtt{new}\ x)P \rangle \longrightarrow_p^e \langle \hat{s}, P\{y/x\} \rangle \quad y \notin f_v(P) \cup b_v(P) \\
(PRESTRICTION) & \langle \hat{s}, (\nu n)P \rangle \longrightarrow_p^e \langle \hat{s}, P\{m/n\} \rangle \qquad m = \mathtt{freshN}(V) \\
(PMATCH) & \langle \hat{s}, [M = M']P \rangle \longrightarrow_p^e \langle \hat{s}\theta, P\theta \rangle \qquad \theta = \mathtt{Mgu}(M, M') \\
(PLCOM) & \dfrac{\langle \hat{s}, P \rangle \longrightarrow_p^e \langle \hat{s}', P' \rangle}{\langle \hat{s}, P \| Q \rangle \longrightarrow_p^e \langle \hat{s}', P' \| Q \rangle} \quad Q' = Q\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } Q' = Q \\
(PRCOM) & \dfrac{\langle \hat{s}, P \rangle \longrightarrow_p^e \langle \hat{s}', Q' \rangle}{\langle \hat{s}, P \| Q \rangle \longrightarrow_p^e \langle \hat{s}', P' \| Q' \rangle} \quad P' = P\theta \text{ if } \hat{s}' = \hat{s}\theta \text{ else } P' = P \\
(PLSUM) & \langle \hat{s}, P + Q \rangle \longrightarrow_p^e \langle \hat{s}, P \rangle \\
(PRSUM) & \langle \hat{s}, P + Q \rangle \longrightarrow_p^e \langle \hat{s}, Q \rangle \\
(PSTR) & \dfrac{P \equiv P' \quad \langle s, P' \rangle \longrightarrow_p^e \langle s', Q' \rangle \quad Q' \equiv Q}{\langle s, P \rangle \longrightarrow_p^e \langle s', Q \rangle}
\end{array}
$$

Figure 7.3: Parametric Transition Rules for the Extended Model

We still have the similar soundness and completeness theorem between extended concrete model and its corresponding parametric model. In the proof, we only list the proof for NEW and $PNEW$ transitions rules, since SUM and $PSUM$ cases are obvious and

other rules are as same as those in proof for Theorem 4.1. A detailed proof can be found in the extended version of [77].

Theorem 7.1 (Soundness and completeness)**.** *Let $\langle \epsilon, P\rangle$ be an initial configuration, and s be a concrete trace. $\langle \epsilon, P\rangle \longrightarrow^{e*} \langle s, P'\rangle$ for some P' if and only if there exists $\hat{s}$, such that $\langle \epsilon, P\rangle \longrightarrow_p^{e*}$ $\langle \hat{s}, P''\rangle$ for some P'', and s is a concretization of $\hat{s}$.*

Proof. "⇒": By an induction on the number of transitions $\longrightarrow^e$ and $\longrightarrow_p^e$, the proof is trivial in the zero-step. We assume in the n-th step the property holds. That is, for each trace s gained in the n-th $\longrightarrow^e$ step, there exists an $\hat{s}$ obtained by the n-th $\longrightarrow_p^e$ step, and $\hat{s}\vartheta = s$ holds for some substitution ϑ from parametric variables to ground messages. Now, we perform a case analysis on the $n+1$ step:

- Case $\langle s, (\texttt{new}\ x)P\rangle$: Then we have $\langle s, (\texttt{new}\ x)P\rangle \longrightarrow^e \langle s, P\{M/x\}\rangle$ and $s \vdash_P M$. Its counterpart configuration is $\langle \hat{s}, (\texttt{new}\ x)P'\rangle$ and $s = \hat{s}(\vartheta \cup \{M/x\})$.

"⇐": By an induction on the number of transitions $\longrightarrow_p^e$ and $\longrightarrow^e$, the proof is trivial in the zero-step. We assume in the n-th step the property holds, that is, for each parametric trace $\hat{s}$ gained by the n-th $\longrightarrow_p^e$ step, if there exists a substitution ϑ from variables to ground messages, and a trace s that satisfies $s = \hat{s}\vartheta$, then s can be obtained by the n-th step of $\longrightarrow^e$. Now, we perform a case analysis on the $n+1$-th step:

- Case $\langle \hat{s}, (\texttt{new}\ x)P\rangle$: If there exists a step in which $\langle \hat{s}, (\texttt{new}\ x)P\rangle \longrightarrow_p^e \langle \hat{s}, P\rangle$, and a ground substitution ϑ where $\hat{s}\vartheta$ is a concrete trace, then $x\vartheta$ is a ground message which can be P-deduced by $s\vartheta$. So $\langle s, (\texttt{new}\ x)P\rangle \longrightarrow^e \langle s, P\{\vartheta(x)/x\}\rangle$, where $P' = P\vartheta$.

□

7.3.2 Refinement Step

From Subsection 4.2, we know that a parametric trace with a rigid message needs to be substituted by unifying a rigid message to the messages in output actions of its prefix parametric traces. In the previous parametric model, the only occasion that a rigid message may occur is in an input action, while in the extended model, such a unification may lead a new inconsistency in the system.

Example 7.1. Consider a naive protocol:

$$\begin{aligned} A \longrightarrow B: &\quad N_A \\ B \longrightarrow A: &\quad \{N_B\}_{-K_B} \end{aligned}$$

Suppose that A may send any possible message to B, which is represented by $(\texttt{new}\, x)\,\overline{a1}\, x.\ \overline{a1}\, x.a2\,\{y\}_{-\mathtt{k}[B]}$ is a parametric trace. In $a2$, a rigid message $\{y\}_{-\mathtt{k}[B]}$ is a requirement of A. After applying unification, the parametric trace deduces to $\overline{a1}\{y\}_{-\mathtt{k}[B]}.a2\{y\}_{-\mathtt{k}[B]}$. But as a principal A, it cannot generate any messages that satisfy the patten $\{y\}_{-\mathtt{k}[B]}$. Thus such kind of parametric messages is also a rigid message, which should be further substituted by applying the similar unification.

Thus, a rigid message in the extended parametric model has the following definition (comparing to Definition 4.3).

Definition 7.4 (Rigid message)**.** Given a parametric trace $\hat{s}$, $\{N\}_L$ in M is a rigid message if

- M is in an input action such that $\hat{s} = \hat{s}'.a(M).\hat{s}''$, and
  - if L is a shared key or a private key, then $\hat{s}' \not\vdash L$ and $\hat{s}' \not\vdash \{N\}_L$;
  - if L is a public key, then there exists some rigid message, or at least one name or binder in N cannot be deduced by the $\hat{s}'$, and $\hat{s}' \not\vdash \{N\}_L$.
- M is in an output action such that $\hat{s} = \hat{s}'.\overline{a}\,M.\hat{s}''$, and
  - $\{N\}_L$ satisfies the above three conditions, and
  - L is not known by the principal that contains the label a.

With the extension of rigid messages, the deductive relation defined in Definition 4.5 also needs to be extended. However, we have the similar results to those in Subsection 4.2. We will prove them to guarantee the correctness.

Definition 7.5 (Deductive relation)**.** Let $\hat{s}$ be a parametric trace such that $\hat{s} = \hat{s}_1.l(M).\hat{s}_2$, in which l is an input or an output label. If there exists a rigid message N in M such that $N \notin el(\hat{s}_1)$, and N is $\hat{\rho}$-unifiable in $\hat{s}_1$, then $\hat{s} \rightsquigarrow^e \hat{s}\hat{\rho}$.

For two parametric traces $\hat{s}$ and $\hat{s}'$, if $\hat{s} \rightsquigarrow^{e*} \hat{s}'$ and there is no $\hat{s}''$ that satisfies $\hat{s}' \rightsquigarrow^e \hat{s}''$, we name $\hat{s}'$ the *normal form* of $\hat{s}$. The set of normal forms of $\hat{s}$ is denoted by $\mathtt{nf}_{\rightsquigarrow^e}(\hat{s})$.

Fact 7.1. *Given a parametric trace $\hat{s}$, $\mathtt{nf}_{\rightsquigarrow^e}(\hat{s})$ is finite.*

A concretization of a parametric trace $\hat{s}$ is still the concretization of $\hat{s}'$ if $\hat{s} \rightsquigarrow^e \hat{s}'$. Thus whether a parametric trace has concretizations is equivalent to whether there exist parametric traces in its $\mathtt{nf}_{\rightsquigarrow^e}(\hat{s})$ that have concretizations.

Lemma 7.2. *If $\hat{s}$ is a parametric trace, and s is a concretization satisfying $s = \hat{s}\vartheta$ where ϑ is a concretized substitution, then $\hat{s}$ is either a normal form, or there exists $\hat{s}'$ such that $\hat{s} \rightsquigarrow^e \hat{s}'$ with $\hat{s}\vartheta = \hat{s}'\vartheta$.*

Proof. If $\hat{s}$ is not a normal form, there exists some rigid message that is not contained in the elementary message set of its prefix. We perform case analysis on the kind of rigid messages $\{N\}_L$.

- If $\{N\}_L$ is an input rigid message in M, where $\hat{s} = \hat{s}'.a(M).\hat{s}''$, then $\{N\}_L \notin el(\hat{s}')$. Since $s = \hat{s}\vartheta$ and s is a trace, $\hat{s}'\vartheta \vdash M\vartheta$. Thus $\{N\}_L\vartheta \in el(\hat{s}'\vartheta)$. By the definition of a rigid message, $L \notin el(\hat{s}')$, and thus $L\vartheta \notin el(\hat{s}')\vartheta$. Since $\{N\}_L\vartheta \in el(\hat{s}'\vartheta) = el(\hat{s}')\vartheta$, there exists $\{N'\}_L \in el(\hat{s}')$ such that $\{N\}_L\vartheta = \{N'\}_L\vartheta$. Thus $\{N\}_L$ and $\{N'\}_L$ are unifiable. Let $\hat{\rho} = \mathtt{Mgu}(\{N\}_L, \{N'\}_L)$, then $\hat{s} \rightsquigarrow^e \hat{s}\hat{\rho}$. Since $\{N\}_L\vartheta = \{N'\}_L\vartheta$, each corresponding parametric variable in two messages will be assigned to the same ground message. Thus, $\hat{s}\vartheta = \hat{s}\hat{\rho}\vartheta$.

- If $\{N\}_L$ is an output rigid message in M, where $\hat{s} = \hat{s}'.\overline{a}\,M.\hat{s}''$. Thus L is not known by the principal P that contains the label a. Since $s = \hat{s}\vartheta$ and s is a trace, and thus $\hat{s}'\vartheta \vdash_P M\vartheta$. L is not known by P, so $\hat{s}'\vartheta \vdash M\vartheta$. Then following the similar proof for the input rigid message case, we can have $\hat{s}\vartheta = \hat{s}\hat{\rho}\vartheta$.

□

Lemma 7.3. *Let $\hat{s}$ be a parametric trace, and let $\hat{s}'$ be a normal form in $\mathtt{nf}_{\rightsquigarrow^e}(\hat{s})$. $\hat{s}'$ has a concretization, if and only if, for each decomposition $\hat{s}' = \hat{s}'_1.l(M).\hat{s}'_2$ in which l is either an input label or an output label, each rigid message N in M satisfies $N \in el(\hat{s}'_1)$.*

Proof. "⇒": Prove by contradiction. Assume a normal form $\hat{s}'$ has concretizations s such that $s = \hat{s}'\vartheta$. If $\hat{s}'$ does not satisfy the first requirement, there exists at least one rigid message $\{N\}_L$ in $\hat{s}'$ that is not $\hat{\rho}$-unifiable in its prefix $\hat{s}'_1$. Thus $\{N\}_L\vartheta \notin el(\hat{s}'_1)\vartheta$. If it is an input rigid message, $\hat{s}'_1\vartheta \not\vdash L$, then $\hat{s}'_1\vartheta \not\vdash \{N\}_L\vartheta$. If it is an output rigid message, $\hat{s}'_1\vartheta \not\vdash_P \{N\}_L\vartheta$, where P is the process containing label a. This contradicts the definition of a trace.

"⇐": Let ϑ be an arbitrary concretized substitution that assigns each parametric variable in $\hat{s}'$ to a name in $\mathcal{E}$, then for each decomposition $\hat{s}'\vartheta = \hat{s}'_1\vartheta.a(M\vartheta).\hat{s}'_2\vartheta$, $\hat{s}'_1\vartheta \vdash M\vartheta$ is satisfiable. Thus $\hat{s}'\vartheta$ is a trace, and also a concretization of $\hat{s}'$. □

A satisfiable normal form is a normal form of $\hat{s}$ that satisfies the requirements in Lemma 7.3. Note that it is different from the definition in Subsection 4.2.

$\mathtt{snf}_{\rightsquigarrow^e}(\hat{s})$ denotes the set of *satisfiable normal forms* of $\hat{s}$. Since $\mathtt{snf}_{\rightsquigarrow^e}(\hat{s}) \subseteq \mathtt{nf}_{\rightsquigarrow^e}(\hat{s})$, $\mathtt{snf}_{\rightsquigarrow^e}(\hat{s})$ is finite.

Fact 7.2. *Given a parametric trace $\hat{s}$, $\mathtt{snf}_{\rightsquigarrow^e}(\hat{s})$ is finite.*

Thus, a parametric trace has a concretization if and only if $\mathtt{snf}_{\rightsquigarrow^e}(\hat{s}) \neq \emptyset$.

Lemma 7.4. *Let $\hat{s}$ be a parametric trace, and let s be a trace. s is a concretization of $\hat{s}$ if and only if s is a concretization of some $\hat{s}'$ with $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow^e}(\hat{s})$.*

Proof. "⇒" If s is a concretization of $\hat{s}$, then there exists a concretized substitution ϑ with $s = \hat{s}\vartheta$. By Lemma 7.2 we can get either $\hat{s}$ is a normal form, or $\hat{s}$ can be deduce to a parametric trace $\hat{s}'$ by $\rightsquigarrow^e$ such that $s = \hat{s}'\vartheta$. If $\hat{s}$ is a normal form and it has a concretization s, so $\hat{s}$ is also a satisfiable normal form according to Lemma 7.3 . If $\hat{s}$ is not a normal form, the number of rigid messages in $\hat{s}$ is finite, so there exists a normal form $\hat{s}''$ of $\hat{s}$ that satisfies $\hat{s}\vartheta = \hat{s}''\vartheta$ by repeatedly applying lemma 7.2. Since $\hat{s}'$ has the concretization s, $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow^e}(\hat{s})$.

"⇐" If s is a concretization of the satisfiable normal form $\hat{s}'$ such that $\hat{s}' \in \mathtt{snf}_{\rightsquigarrow^e}(\hat{s})$, we have $s = \hat{s}'\vartheta$ for some concretized substitution ϑ. $\hat{s}'$ is a normal form of $\hat{s}$, so $\hat{s}' = \hat{s}\hat{\rho}$ for some $\hat{\rho}$, in which $s = \hat{s}'\vartheta = \hat{s}\hat{\rho}\vartheta$. Thus s is a concretization of $\hat{s}$. □

Theorem 7.5. *A parametric trace $\hat{s}$ has a concretization if and only if $\mathtt{snf}_{\rightsquigarrow^e}(\hat{s}) \neq \emptyset$.*

The theorem is a corollary of Lemma 7.4.

Although the definitions and theorems above have some differences in detail from definitions and theorems in Subsection 4.2, it is still decidable to check which parametric trace has its corresponding concretizations in the extended parametric model. Thus an action term can be checked in this parametric model, by the similar way in Subsection 5.2.

7.4 Attacks of the Simplified ZG Protocol and Its Modification

For the simplified Zhou-Gollmann fair non-repudiation protocol, counterexamples that violate NRO, FAIRO and FAIRM are found in parametric traces. One for NRO is shown as follows:

$$\overline{a1}\,\{B, N_A, x_2\}_{-\mathtt{k}[A]}.b1(\{B, N_A, x_2\}_{-\mathtt{k}[A]}).\overline{b2}\,\{A, y_5\}_{-\mathtt{k}[B]}.s1(\{B, N_A, x_2\}_{-\mathtt{k}[A]}).$$
$$\overline{s2}\,\{A, B, N_A, x_2\}_{-\mathtt{k}[A]}.\overline{s3}\{A, B, N_A, x_2\}_{-\mathtt{k}[A]}.b3(\{A, B, N_A, x_2\}_{-\mathtt{k}[A]}).$$
$$\overline{evid}_A\,(\{B, N_A, x_2\}_{-\mathtt{k}[A]}, \{A, B, N_A, x_2\}_{-\mathtt{k}[A]})$$

It actually represents the following replay attack, A sent a message in the flow, then aborted the protocol. After that, an intruder (or B) pretended A to continue the protocol, sending the used message to S.

$$\begin{array}{rcll} A & \longrightarrow & B: & \{B, N_A, x\}_{-K_A} \\ B & \longrightarrow & A: & \{A, N_A, x\}_{-K_B} \\ I(A) & \longrightarrow & S: & \{B, N_A, x\}_{-K_A} \\ S & \longrightarrow & I(A): & \{A, B, N_A, x\}_{-K_S} \\ S & \longrightarrow & B: & \{A, B, N_A, x\}_{-K_S} \end{array}$$

For FAIRO, the same counterexample is also detected.

A comparison is the full ZG protocol, which guarantees all five properties. The full ZG protocol is described in [116]. It introduces a set of public names, named *flags* to identify the steps of the protocol. They can prevent replay attacks for the simplified ZG protocol. For formal definitions of the properties of full ZG protocol, refer to [103].

The full ZG protocol is given informally as follows.

$$\begin{array}{rcll} A & \longrightarrow & B: & \{F_{NRO}, B, N_A, \{M\}_K\}_{-K_A} \\ B & \longrightarrow & A: & \{F_{NRR}, A, N_A, \{M\}_K\}_{-K_B} \\ A & \longrightarrow & S: & \{F_{SUB}, B, N_A, K\}_{-K_A} \\ S & \longrightarrow & A: & \{F_{CON}, A, B, N_A, K\}_{-K_S} \\ S & \longrightarrow & B: & \{F_{CON}, A, B, N_A, K\}_{-K_S} \end{array}$$

Chapter 8

Protocol-Independent Security Specifications

Most formal methods for security protocols, especially process calculi based approaches, mixed the description of a security protocol from the specification for a given security property. They gave a specification for a security property by inserting some facilities to define the property manually [80, 28, 102, 100]. This chapter shows how to construct the specification for a specific security property from the description of a security protocol automatically.

As we know, it is not sufficient for defining security properties on the description of a security protocol. Processes that describe behaviors for representations of a given security property need to be inserted in the description of a security protocol, in order to define this security property. These behaviors do not belong to the prescription of a protocol. For instance, to represent the secrecy property, a behavior that "someone can obtain the message from the network" is needed; to represent the authentication property, a behavior that "A principal thinks the message comes from the other principal" is needed, etc.

A *probing process* is a process transformed from a formal description of a security protocol, with the aim of representing security properties. Basically, there are two kinds of transformations to a probing process, *declaration process insertions* and *guardian process compositions*.

A declaration process insertion replaces some occurrences of $\mathbf{0}$ to a process with only an output action, $\overline{c}\,M.\mathbf{0}$, named a *declaration process*. For instance,

$$a(x).[x = b]\,\mathbf{0} \qquad \Longrightarrow \qquad a(x).[x = b]\,\overline{c}\,x.\mathbf{0}$$

Variables in the message M of the action $\overline{c}\,M$ are free, while a probing process is closed. Thus all free variables in M should be bound by some binding operators (like $a(x)$ in the example). Intuitively, a declaration process describes that a principal provides a message it received, after it validates the message. In the above example, a principal received a message trough x, and checked whether it is equal to b. If so, it provides the validated message by the output action $\overline{c}\,x$.

A guardian process composition composes a formal description of a protocol to a process with only an input action, $c(x).\mathbf{0}$, named a *guardian process*. For instance,

$$P \qquad \Longrightarrow \qquad P \| c(x).\mathbf{0}$$

The variable x is bound in the input action $c(x)$. Intuitively, a guardian process is similar to a fresh principal inserted in the network, listening all messages leaked in the network, and trying to find out whether confidential messages are leaked.

By these two transformations, most of security properties can be defined by action terms introduced in Chapter 5. A *syntax scanners* are needed for a given security property when a transformation is preformed. For instance, to find out which occurrences of $\mathbf{0}$ will be substituted by a declaration process in a declaration process insertion. This chapter shows how to automatically define the secrecy and authentication properties when a protocol description is given.

8.1 Security Specification Transformations

Given a process P, the *context* $P[.]$ is obtained when all occurrences of $\mathbf{0}$ in P are replaced by *holes*, $[.]$. Let $\phi(P)$ be the set of holes in $P[.]$.

Definition 8.1 (Declaration process insertion)**.** Let $P[.]$ be the context of P. Given a set $\psi \subseteq \phi(P)$, and a message M, $P_{\psi,M}$ is a probing process generated from P, such that holes in ψ are inserted by the same process $\overline{c}\,M.\mathbf{0}$ with a fresh label $\overline{c}$ (named a *declaration process*), and holes in $\phi(P) - \psi$ are inserted by $\mathbf{0}$ in $P[.]$.

For a given security property, the choice of ψ and M in a declaration process insertion are not independent to the protocol description. A syntax scanner is performed to decide ψ when performing a declaration process insertion. We will illustrate it when defining the authentication property.

Definition 8.2 (Guardian process composition)**.** A probing process P_g is P composed with a process $c(x).\mathbf{0}$ with a fresh label c (named a *guardian process*), that is, $P_g \triangleq P \| c(x).\mathbf{0}$.

Definition 8.3 (Probing transformations)**.** Given a process P that describes a protocol, a probing transformation is generated from P, by applying the two transformations above finitely many times, and returns a process (named a *probing process*).

Remark 8.1. Intuitively, a probing process transformed from a protocol description is used to represent security properties. Declaration process insertion is used to show that a principal can provide some message M after validating it, used for authentication, non-repudiation, and fairness. Guardian process composition is used to check whether a message is observable in the environment, for secrecy and fairness.

8.2 Syntax Tree of a Process

For a given process, a *syntax (binary) tree* is a finite, labeled, directed (binary) tree, where the internal nodes are labeled by operators, and the leaf nodes are $\mathbf{0}$ and identifiers that occurs the second time in the tree. In the latter case, the process can also be represented as a graph. We just omit its back edges here. Let's take several examples.

Example 8.1. The syntax tree of the process

$$a1(x).\overline{a2}\,M.\mathbf{0}$$

is represented as (a) in Figure 10.2.

Example 8.2. The syntax tree of the process

$$a1(x).\overline{a2}\,M.\mathbf{0} \| b1(x).\ let\ (y,z) = x\ in\ \mathbf{0}$$

is represented as (b) in Figure 10.2.

Example 8.3. The syntax tree of the process

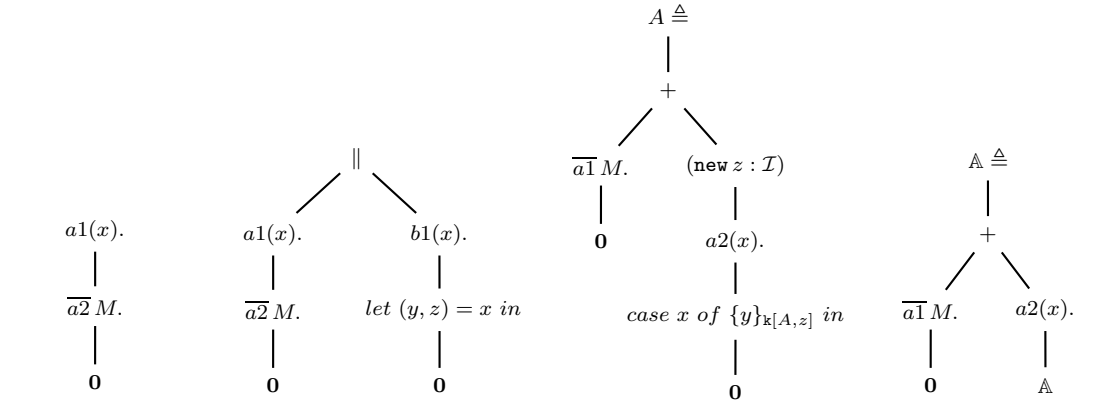
$$A \triangleq \overline{a1}\,M.\mathbf{0} + (\mathtt{new}\, z : \mathcal{I})a2(x).\ case\ x\ of\ \{y\}_{\mathtt{k}[A,z]}\ in\ \mathbf{0}$$

is represented as (c) in Figure 10.2.

Example 8.4. The syntax tree of the process

$$\mathbb{A} \triangleq \overline{a1}\,M.\mathbf{0} + a2(x).\mathbb{A}$$

is represented as (d) in Figure 10.2.



(a) Example 7.1 (b) Example 7.2 (c) Example 7.3 (d) Example 7.4

Figure 8.1: Example Figures for Syntax Trees of Processes

Representing a process by a syntax binary tree provides facilities for performing syntax scanning. In the latter subsections of this chapter, we assume all processes are represented by syntax binary trees.

8.3 Secrecy

8.3.1 General Secrecy Definition

Intuitively, the secrecy property means that the environment should never learn a confidential datum two principals shared. Thus, to define the secrecy property needs four parameters, the protocol description (denoted by x_p), the confidential message (denoted by x_m), two principals who share the message (denoted by x_s for its sender, and x_r for its receiver, respectively). We have the following definition,

Definition 8.4 (Secrecy property)**.** Given a protocol description x_p, a confidential message x_m, two principals who share the x_m, x_s and x_r, the secrecy property of the protocol is defined as follows,

$$\texttt{Secrecy}(x_p, x_m, x_s, x_r) = \langle \epsilon, \texttt{Probing\_s}(x_p, x_m, x_s, x_r)\rangle \models \neg check(\texttt{BN}(x_m)[x_s, x_r])$$

where `Probing_s` returns its corresponding probing process for the secrecy property. $\texttt{BN} : \mathcal{N} \rightarrow \mathcal{B}$ maps a name to a binder name that does not occur in x_p.

8.3.2 Generating the Secrecy Specification

In the `Probing_s`, the first and main task is to replace the confidential message x_m in x_p to a binder. This task is performed by a syntax scanner

$$\texttt{Scanner\_s}(x_p, x_m, x_s, x_r)$$

The second task is to perform a guardian process insertion, composing a new process $check(x).\mathbf{0}$ to the process x_p.

In essence, $\texttt{Scanner\_s}(x_p, x_m, x_s, x_r)$ is a pre-order, depth-first search on the syntax binary tree of x_p. It gathers parameters' information of the binder that replaces x_m, as follows:

- The A in the nearest ancestor that contains $A \triangleq$ of the node containing x_m is the first parameter of the binder that replaces x_m (denotes its sender). It should be equal to x_s; Otherwise the description x_p has mistakes.
- The x in the nearest ancestor that contains $(\texttt{new}\, x : \mathcal{I})$ of the node containing x_m is the second parameter of the binder that replaces x_m (denotes its receiver).
- If no ancestors of the node containing x_m contains $(\texttt{new}\, x : \mathcal{I})$, the second parameter of the binder is x_r.

The algorithms of `Probing_s` and `Scanner_s` are defined in Algorithm 1.

8.3.3 Examples for Generating Secrecy Specifications

The AG Protocol

The description of the AG protocol (see Subsection 3.3.1) is as follows,

$$\begin{aligned}
A \triangleq & (\texttt{new}\ x : \mathcal{I})(\nu\, M)\overline{a1}(A, \{x, \texttt{k}[A, x]\}_{\texttt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\texttt{k}[A,x]}).\mathbf{0} \\
B \triangleq & b1(x).case\ x\ of\ \{x'\}_{\texttt{k}[B,S]}\ in\ let\ (y, z) = x'\ in\ [y = A]\, b2(w).\ let\ (w', w'') = w \\
& in\ [w' = A]\ case\ w''\ of\ \{u\}_z\ in\ let\ (u', u'') = u\ in\ [u' = A]\, \mathbf{0} \\
S \triangleq & s1(x).let\ (y, z) = x\ in\ case\ z\ of\ \{u\}_{\texttt{k}[y,S]}\ in\ let\ (u', u'') = u\ in\ \overline{s2}\{y, u''\}_{\texttt{k}[u',S]}.\mathbf{0} \\
SYS^{AG} \triangleq & A \| S \| B
\end{aligned}$$

When performing $\texttt{Scanner\_s}(SYS^{AG}, M, A, B)$, since $A \triangleq$ is the nearest ancestor containing the operator $\triangleq$, and thus the first parameter of the binder is A; $(\texttt{new}\ x : \mathcal{I})$ is

Algorithm 1 Syntax Scanner and Probing Procedure for Secrecy Property

function PROBING_S(x_p, x_m, x_s, x_r)
  return SCANNER_S(x_p, x_m, x_s, x_r)$\|check(x).\mathbf{0}$
end function

function SCANNER_S(x_p, x_m, x_s, x_r)
  $s \leftarrow nil$ ▷ records the sender of x_m
  $r \leftarrow nil$ ▷ records the expected receiver of x_m
  return SCANNING_S (x_p, x_m, s, r, x_s, x_r)
end function

function SCANNING_S(x_p, x_m, s, r, x_s, x_r)
  if $x_p = (A \triangleq)$ **then**
    $s \leftarrow A$
  end if
  if $x_p = (\mathtt{new}\, x : \mathcal{I})$ **then**
    $r \leftarrow x$
  end if
  if x_m in x_p **then**
    Substituting x_m in x_p to BINDERGEN(x_m, s, r, x_s, x_r)
  end if
  SCANNING_S ($x_p.lchild$, x_m, s, r, x_s, x_r)
  SCANNING_S ($x_p.rchild$, x_m, s, r, x_s, x_r)
  return x_p
end function

function BINDERGEN(x_m, s, r, x_s, x_r)
  if $s! = x_s$ **then** ▷ if the sender is not the same as declared
    Raise Error
  else
    if $r = nil$ **then** ▷ if no expected receiver, it will be the declared one
   return $\mathtt{BN}(x_m)[s, x_r]$
    else
   return $\mathtt{BN}(x_m)[s, r]$
    end if
  end if
end function

also the nearest ancestor containing the primitive `new`, and thus the second parameter of the binder is x. So after performing the algorithm, process A in SYS^{AG} will be replaced to

$$A \triangleq (\texttt{new}\ x : \mathcal{I})(\nu\, \texttt{m}[A, x])\overline{a1}(A, \{x, \texttt{k}[A, x]\}_{\texttt{k}[A,S]}).\overline{a2}(A, \{A, \texttt{m}[A, x]\}_{\texttt{k}[A,x]}).\mathbf{0}$$

This is the exact $SYS^{AG'}$ defined in Subsection 5.3.1.

Since the result of $\texttt{Probing\_s}(SYS^{AG}, M, A, B)$ is the SYS_s^{AG} defined in Subsection 5.3.1, the secrecy for the AG protocol

$$\texttt{Secrecy}(SYS^{AG}, M, A, B) = \langle \epsilon, SYS_s^{AG} \rangle \models \neg check(\texttt{m}[A, B])$$

is same as the Characterization 5.1.

The NSPK Protocol

The description of the NSPK protocol (see Subsection 3.3.2) is as follows,

$$\begin{aligned}
A \triangleq & (\texttt{new}\ x_a : \mathcal{I})(\nu\ N_A)\overline{a1}\{A, N_A\}_{+\texttt{k}[x_a]}.a2(y_a).\ case\ y_a\ of\ \{y_a'\}_{-\texttt{k}[A]}\ in \\
& let\ (z_a, z_a') = y_a'\ in\ [z_a = N_A]\,\overline{a3}\{z_a'\}_{+\texttt{k}[x_a]}.\mathbf{0} \\
B \triangleq & (\nu\ N_B)\, b1(x_b).case\ x_b\ of\{x_b'\}_{-\texttt{k}[B]}\ in\ let\ (y_b, y_b') = x_b'\ in\ [y_b = A] \\
& \overline{b2}\{y_b', N_B\}_{+\texttt{k}[A]}.b3(z_b).\ case\ z_b\ of\ \{u_b\}_{-\texttt{k}[B]}\ in \\
& [u_b = N_B]\,\mathbf{0} \\
SYS^{NSPK} \triangleq & A \| B
\end{aligned}$$

When performing $\texttt{Scanner\_s}(SYS^{AG}, N_B, B, A)$, since $B \triangleq$ is the nearest ancestor containing $\triangleq$, and thus the first parameter of the binder is B; there are no ancestors of the node with N_B that contain `new`, and thus the second parameter of the binder is equal to A. So after performing the algorithm, B in SYS^{NSPK} will be replaced to

$$\begin{aligned}
B_s \triangleq & (\nu\, \texttt{N}_\texttt{B}[B, A])b1(x_b).case\ x_b\ of\{x_b'\}_{-\texttt{k}[B]}\ in\ let\ (y_b, y_b') = x_b'\ in\ [y_b = A] \\
& \overline{b2}\{y_b', \texttt{N}_\texttt{B}[B, A]\}_{+\texttt{k}[A]}.b3(z_b).\ case\ z_b\ of\ \{u_b\}_{-\texttt{k}[B]}\ in \\
& [u_b = \texttt{N}_\texttt{B}[B, A]]\,\mathbf{0}
\end{aligned}$$

Since the result of $\texttt{Probing\_s}(SYS^{NSPK}, N_B, B, A)$ is exact the SYS_s^{NSPK} defined in Subsection 5.3.2, the secrecy for the NSPK protocol

$$\texttt{Secrecy}(SYS^{NSPK}, N_B, B, A) = \langle \epsilon, SYS_s^{NSPK} \rangle \models \neg check(\texttt{N}_\texttt{B}[B, A])$$

is the same as the Characterization 5.3.

8.4 Authentication

8.4.1 General Authentication Definition

Intuitively, the authentication property means that a message that purports to be sent from a certain principal was indeed originated by that principal. Usually, we concern

whether a flow is satisfied authentication in a given security protocol. For instance, the third flow of the AG protocol (see Subsection 3.3.1) is checked whether satisfies authentication. In the NSPK protocol (see Subsection 3.3.2), it is the third flow that analyzed. In the multiple WL protocol (see Subsection 3.3.3), it is also the third flow that analyzed.

When we focus the formal protocol description in one session, a flow is regarded as a pair of labels. For instance, the third flow in the AG protocol is represented as a pair of labels $(a2, b2)$, meaning that the flow is sent by A in the action labeled by $a2$, and received by B in the action labeled by $b2$. Similarly, the third flow in the NSPK protocol is represented as $(a3, b3)$. Sometimes in multiple sessions, the sending label of a concerned flow is nondeterministic, which is represented as a disjunction of finite labels. For instance, in the multiple WL protocol the concerned flow can be represented as $(a3 \vee a'2, b3)$.

Therefore, to define the authentication property, three parameters are needed: the protocol description (denoted by x_p), a disjunction of labels for sending actions (denoted by $\bigvee_{i\in n} ls_i$), and a label lr for the receiving action. We have the following definition,

Definition 8.5 (Authentication property)**.** Given a protocol description x_p, a disjunction of labels for sending actions $\bigvee_{i\in n} ls_i$, and a label lr for the receiving action, the authentication property for the protocol is defined as follows,

$$\texttt{Authentication}(x_p, \bigvee_{i\in n} ls_i, lr) = \langle \epsilon, \texttt{Probing\_a}(x_p, lr)\rangle \models \bigvee_{i\in n}(ls_i\, x) \hookleftarrow \overline{acc}\, x$$

where `Probing_a` returns its corresponding probing process for authentication.

8.4.2 Generating the Authentication Specifications

In the $\texttt{Probing\_a}(x_p, lr)$, $\overline{acc}$ is the generated label when the declaration process insertion is performed. During the declaration process insertion, the message attached by $\overline{acc}$ is the same message attached by lr (this message is a bound variable). The process $\overline{acc}\, M.\mathbf{0}$ will replace all occurrences of $\mathbf{0}$ in the syntax sub-tree with "$lr(M)$." as its root.

A syntax scanner, $\texttt{Scanner\_a}(x_p, lr)$ is performed to returns the node that contains label lr in x_p, and to return the message m_r attached by lr. Then by an algorithm $\texttt{Insert\_Acc}(x_p, m_r)$, all occurrences of $\mathbf{0}$ in the sub-tree whose root containing lr are replaced by the process $\overline{acc}\, m_r.\mathbf{0}$.

The algorithms are defined in Algorithm 2.

8.4.3 Examples for Generating Authentication Specifications

The NSPK Protocol

The authentication of the NSPK protocol is defined as follows,

$$\texttt{Authentication}(SYS^{NSPK}, a3, b3) = \langle \epsilon, \texttt{Probing\_a}(SYS^{NSPK}, b3)\rangle \models a3\, x \hookleftarrow \overline{acc}\, x$$

In $\texttt{Probing\_a}(SYS^{NSPK}, b3)$, when $\texttt{Scanner\_a}(SYS^{NSPK}, b3)$ is applied, it returns a sub-tree with $b3(z_b)$ as its root, and z_b. Then by $\texttt{Insert\_Acc}(b3(z_b), z_b)$, all $\mathbf{0}$ in the sub-tree is replaced by $\overline{acc}\, z_b$ (It is the same specification defined in Subsection 5.3.2).

Algorithm 2 Syntax Scanner and Probing Procedure for Authentication Property

```
function PROBING_A(x_p, lr)
    INSERT_ACC(SCANNER_A(x_p, lr))
     return x_p
end function

function SCANNER_A(x_p, lr)
    if x_p = lr M then
     return (x_p, M)
    else
        SCANNER_A (x_t.lchild, lr)
        SCANNER_A (x_t.rchild, lr)
    end if
end function

function INSERT_ACC(x_p, m_r)
    if x_p = 0 then
        x_p ← acc̄ m_r.0
    else
        INSERT_ACC (x_t.lchild, m_r)
        INSERT_ACC (x_t.rchild, m_r)
     return x_p
    end if
end function
```

The Multiple WL Protocol

The description of the WL protocol in multiple sessions (see Subsection 3.3.3) is as follows,

$$
\begin{aligned}
A^{(2)} \triangleq & \overline{a1}\, A.a2(x_a).\overline{a3}\,\{x_a\}_{\mathtt{k}[A,S]}.\mathbf{0} \| \overline{a'1}\, A.a'2(x'_a).\overline{a'3}\,\{x'_a\}_{\mathtt{k}[A,S]}.\mathbf{0} \\
B^{(2)} \triangleq & (\nu\ N_B)\, b1(x_b).[x_b = A]\,\overline{b2}\, N_B.b3(y_b).\overline{b4}\,(B, \{x_b, y_b\}_{\mathtt{k}[B,S]}).b5(z_b).case\ z_b \\
& of\ \{u_b\}_{\mathtt{k}[B,S]}\ in\ [u_b = N_B]\,\mathbf{0} \| (\nu\ N'_B) b'1(x'_b).\overline{b'2}\, N'_B.b3(y'_b). \\
& \overline{b4}\,(B, \{x'_b, y'_b\}_{\mathtt{k}[B,S]}).b5(z'_b).\ case\ z'_b\ of\ \{u'_b\}_{\mathtt{k}[B,S]}\ in\ [u'_b = N'_B]\,\mathbf{0} \\
S \triangleq & s1(x_s).let\ (x'_s, x''_s) = x_s\ in\ case\ x''_s\ of\ \{y_s\}_{\mathtt{k}[x'_s,S]}\ in\ let\ (z_s, w_s) = y_s\ in \\
& case\ w_s\ of\ \{u_s\}_{\mathtt{k}[z_s,S]}\ in\ \overline{s2}\,\{u_s\}_{\mathtt{k}[x'_s,S]}.\mathbf{0} \\
S^{(2)} \triangleq & S \| S \\
SYS^{(2)} \triangleq & A^{(2)} \| S^{(2)} \| B^{(2)}
\end{aligned}
$$

The authentication of the multiple WL protocol is defined as follows,

$$\mathtt{Authentication}(SYS^{(2)}, a2 \vee a'2, b3) = \langle \epsilon, \mathtt{Probing\_a}(SYS^{(2)}, b3) \rangle \models a2\, x \vee a'2\, x \hookleftarrow \overline{acc}\, x$$

In $\mathtt{Probing\_a}(SYS^{(2)}, b3)$, when $\mathtt{Scanner\_a}(SYS^{(2)}, b3)$ is applied, it will return a sub-tree with $b3(y_b)$ as its root, and y_b. Then by $\mathtt{Insert\_Acc}(b3(y_b), y_b)$, all $\mathbf{0}$ in the sub-tree is replaced by $\overline{acc}\, y_b$ (It is the same specification defined in Subsection 5.3.3).

8.5 Other Properties

Other properties, such as non-repudiation and fairness, are relied heavily on human guidance for their formal definitions. The objectives of these properties are relationships of messages in evidences each principal accepts, and these evidences vary from different protocols. Although probing processes for these properties can be obtained by the similar automatic way for the authentication property, yet we cannot define the properties by just performing syntax analysis on processes.

Chapter 9

Implementation Issues and Experimental Results

We implement the model checking method on the three parametric models introduced in Chapter 5, 6, and 7 by Maude [40], a language and system supporting both equational and rewriting logic computation for a wide range of applications. Maude describes model generation rules by rewriting, instead of describing a model directly, such that each property can be checked at the same time while a model is being generated. It is therefore named an *on-the-fly model checking* method.

The basic units of Maude specifications are modules. In Core Maude, there are two kinds of modules: *functional modules* and *system modules*. Functional modules define data types and operations on them by means of equational theories, whose equations are assumed to be confluent and terminating. System modules specify a model by a rewrite theory, and the model is a transition system with an initial term. For a finite system, Maude `search` command explores all possible execution paths from the initial term for reachable states satisfying some property.

A basic functional module mainly has four parts: sorts, operations, variables and equations. Maude can define a sort or several sorts each time, with the key words `sort` and `sorts` respectively. Variables are declared with the key words `var` or `vars`. The key word of operation is `op`. There are two uses of operations: as a constructor of a sort, and as a declaration of a function. `[ctor]` is a key attribute of a constructor. A function can be implemented by a set of equations, with the key words `eq` and `ceq` (conditional equation). The use of variables in equations does not carry actual values. Rather, they stand for any instance of a certain sort.

Anything defined in a function module can be defined in a system module in the same way. Also, a system module can define a transition system by a set of rewrite laws, whose key words are `rl` and `crl` (conditional rewrite law).

We implement each basic definition and function of the three parametric models by functional modules, and implement a *trace generating system* (for the parametric model in Chapter 6, the trace generating system is actually the pushdown system) using a system module. Then we use `search` command to find whether the negations of the specifications are reachable.

9.1 The Construction of Implementations

9.1.1 Parametric Processes

For efficiency, we adopt parametric processes, instead of original processes. A parametric process is generated from an original process by applying a type inference to the original process. The type system infers information of each bound variable in a process by a statical scanning on the process. We mark each sub-expression whose type is a type variable with a fresh variable. Thus a process can be translated into a parametric process (for the details of the type system and the translation algorithm, see Appendix C).

The set of parametric traces generated by a parametric process is a proper subset of that generated by its original process. Such a compaction actually does not affect the result of checking a given security property (for the detailed discussions, please refer to Subsection 5.6 and Appendix C).

A general parametric process definition is as follows.

Definition 9.1 (Parametric Processes)**.** Let $\hat{\mathcal{P}}$ be a countable set of parametric processes, which is indicated by $\hat{P}, \hat{Q}, \hat{R}, \ldots$. The syntax of processes is defined as follows,

$$\begin{aligned}\hat{P} ::= \; & \mathbf{0} \mid \overline{a}M.\hat{P} \mid a(M).\hat{P} \mid [M = N]\,\hat{P} \mid (\mathtt{new}\ x : \mathcal{A})\hat{P} \mid (\nu\, n)\hat{P} \\ & \mid \mathit{let}\ (M, N) = L\ \mathit{in}\ \hat{P} \mid \mathit{case}\ M\ \mathit{of}\ \{N\}_L\ \mathit{in}\ \hat{P} \\ & \mid \hat{P} + \hat{Q} \mid \hat{P} \| \hat{Q} \mid \hat{P}; \hat{Q} \mid \hat{\mathbb{A}}(\tilde{pr})\end{aligned}$$

where M, N, L are messages defined in Definition 2.1.

Example 9.1. The process of the Abadi-Gordon protocol for authentication, SYS_a^{AG} (described in Subsection 5.3.1), will be translated into the following parametric processes.

$$\begin{aligned}\hat{A}_p \triangleq & (\mathtt{new}\ x_1 : \mathcal{I})(\nu\, M)\overline{a1}(A, \{x_1, \mathtt{k}[A, x_1]\}_{\mathtt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\mathtt{k}[A,x_1]}).\mathbf{0} \\ \hat{B}_p \triangleq & b1(\{A, z_1\}_{\mathtt{k}[B,S]}).\ \mathit{case}\ \{A, z_1\}_{\mathtt{k}[B,S]}\ \mathit{of}\ \{A, z_1\}_{\mathtt{k}[B,S]}\ \mathit{in}\ \mathit{let}\ (A, z_1) = \\ & (A, z_1)\ \mathit{in}\ [A = A]\, b2(A, \{A, w_1''\}_{t_1}).\ \mathit{let}\ (A, (A, \{A, w_1''\}_{t_1})) = \\ & (A, (A, \{A, w_1''\}_{t_1}))\ \mathit{in}\ [A = A]\ \mathit{case}\ \{A, w_1''\}_{t_1}\ \mathit{of}\ \{A, w_1''\}_{z_1} \\ & \mathit{in}\ \mathit{let}\ (A, w_1'') = (A, w_1'')\ \mathit{in}\ [A = A]\, \overline{acc}\, (A, \{A, w_1''\}_{t_1}).\mathbf{0} \\ \hat{S}_p \triangleq & s1(x, \{y, z\}_{\mathtt{k}[x_k,S]}).\mathit{let}\ (x, \{y, z\}_{\mathtt{k}[x_k,S]}) = (x, \{y, z\}_{\mathtt{k}[x_k,S]})\ \mathit{in}\ \mathit{case}\ \{y, z\}_{\mathtt{k}[x_k,S]} \\ & \mathit{of}\ \{y, z\}_{\mathtt{k}[x,S]}\ \mathit{in}\ \mathit{let}\ (y, z) = (y, z)\ \mathit{in}\ \overline{s2}\{x, z\}_{\mathtt{k}[y,S]}.\mathbf{0} \\ \hat{SYS}_p^{AG} \triangleq & A_p \| S_p \| B_p\end{aligned}$$

From the example introduced above, we can see that except input and output actions, other actions, such as pair splitting, decrypting, and match, are all redundant action essentially. So $\hat{B}_p$ can be simplified as

$$\hat{B}_p \triangleq b1(\{A, z_1\}_{\mathtt{k}[B,S]}).b2(A, \{A, w_1''\}_{t_1}).\overline{acc}\, (A, \{A, w_1''\}_{t_1}).\mathbf{0}$$

Thus parametric traces generated by $\hat{SYS}_p^{AG}$ is rather smaller than the original one.

Remark 9.1. Furthermore, each bounded variable and name is explicitly presupposed to be distinguished from each other by α-conversions [101]. Thus binding operations such as ν (for names) and `new` (for variables) are also avoided in the implementation.

9.1.2 Sorts in Implementations

For the parametric systems, there are slight differences between implementing a symmetric key protocol system and an asymmetric key protocol system, according to Definitions 4.3 and 7.4, a symmetric key rigid message is "context-free", while an asymmetric key one, which is decided in a context of parametric messages, is "context-sensitive". In Maude, the functions that judge whether a message is a rigid message are declared as follows:

```
op isSharedRigid : Message -> Bool .
op isPublicRigid : Message Messagelist -> Bool .
```

This subsection introduces sorts of structures and main functions in implementations based mainly on a symmetric-key protocol system. The implementation can be naturally encoded in a asymmetric-key system, which has also been implemented.

In parametric models, types and constructors of the parametric message, the parametric action (which is a 3-tuple consisting of a label, an input/output signal, and a parametric message) and the parametric trace (which is a list of actions) are defined in their function modules, named `MESSAGE`, `ACTION`, and `TRACE` respectively, as follows.

```
sort Message .
  op  name : Nat -> Message [ ctor ] .
  op   px  : Nat -> Message [ ctor ] .
  op k[_,_] : Message Message -> Message [ ctor prec 15 ] .
  op (_,_) : Message Message -> Message [ ctor ] .
  op {_}_  : Message Message -> Message [ ctor prec 20 ] .
  op Hash()   : Message -> Message [ Prec 20 ] .


sort Action .
  op  <_,_,_> : Label IO Message -> Action [ ctor prec 23 ] .

sort Trace .
  subsort Action < Trace .
  op Nil : -> Trace [ctor] .
  op _._ : Trace Trace -> Trace [ ctor assoc id: Nil prec 25 ] .
```

The main function of implementations is the refine step, the trace deductive relation, $\rightsquigarrow$ defined in Definition 4.5 (resp. $\rightsquigarrow^e$, defined in Definition 7.5 for non-repudiation and fairness). It accepts a parametric trace, and returns a new generated parametric trace. Thus its sorts is defined as follows,

```
op RefinementStep : Trace -> Trace .
```

In the refinement step, there are three main functions.

- Firstly, search the given parametric trace, finding its first rigid message, and returning the rigid message and a message set of messages in its prefix parametric trace.

- Secondly, decompose each message in the message set to its atomic messages (see Definition 4.4).
- Lastly, try to unify the rigid message with each message in the atomic message set. returns all possible substitution results.

The first function accepts a parametric trace and a rigid message list, as an environment of the parametric trace, and returns the trace's first rigid message, a boolean (true if there exists some rigid message in the trace), and a message list (as an elementary message list ready for unifying the rigid message). The basic strategy is, the function will return the first rigid message and its elementary message list. Maude does not allow the definition of product sort, so we need to define a `Anares` sort, which is a 3-tuple of `Message`, `Bool` and `Messagelist`. The Sort of the function, named `AnalyzingTrace`, is defined as follows.

```
sort Anares .
  op [_,_,_] : Message Bool Messagelist -> Anares .
  op getMessage : Anares -> Message .
  op getBool    : Anares -> Bool .
  op getMessagelist : Anares -> Messagelist .

  op AnalyzingTrace : Trace Messagelist -> Anares .
```

Remark 9.2. Due to the differences of rigid message definitions (in Definition 4.3 and Definition 7.4, respectively) and satisfied normal forms in the model for the analysis of authentication and security properties, and for the analysis of non-repudiation and fairness properties, the implementations of this function in two parametric systems are different.

The second function is simple, it accepts a message list and returns a generated message list.

```
op elementary : Messagelist -> Messagelist .
```

For the unification function, we adopt the simplest implementation, including an occurrence check.

```
sort Result .
  op (_,_) : Substitutions Bool -> Result [ctor] .
  op getSubstitution : Result -> Substitutions .
  op getBool : Result -> Bool .

  op oCheck  : Message Message -> Bool .
  op unifying : Message Message -> Result [ comm ] .
```

For the details of these functions, see Subsection **??** in Appendix D.

9.1.3 Trace Generating System

A *trace generating system* is embedded in a system module. In the trace generating system, there are two kinds of trace generating rules. The first kind of rules comes from

the parametric transition relations. These rules are protocol specific rules for a given protocol. We name a parametric trace generated by these rules an *original trace*. We will illustrate these rules by examples, as one fragment of the protocol description for three models in the Subsection 8.2, 8.3, and 8.4 respectively.

The second kind of rules is a common part of each protocol in bounded sessions, which generates new parametric trace by the refinement step, These rules deduce a parametric trace to a new one by applying a substitution, which is the result of unifying a rigid message to an elementary message in its prefix parametric trace. We name a parametric trace generated by these rules, and needed to be further substituted, a *pending trace*. Furthermore, we name a parametric trace which is a satisfiable normal form of some original trace a *satisfiable trace*.

We define the state of the trace generating system as a 3-tuple, $\langle tr, S, k\rangle$, where

- tr is a parametric trace.
- S is a list of substitutions.
- k is a type of tr, where $k \in \{ot, st, pt\}$. ot denotes an original trace, st denotes a satisfiable trace and pt represents a pending trace.

In Maude, the state is defined as follows:

```
sort Tracestate Tracetype State .
  ops  ot st pt : -> Tracetype [ctor] .
  op [_]  : Trace -> Tracestate [frozen] .
  op <_,_,_> : Tracestate Substitutionlist Tracetype -> State .
```

Let's specify how a type of a parametric trace changed in our trace generating system: If a parametric trace is labeled pt, and its substitution list is not empty, it will be deduced to a new parametric trace by applying the first substitution in its substitution list. At the same time, a new substitution list of the new trace will be calculated dynamically by applying `analyzingTrace` to the new trace. Furthermore, given a state, we also shrink its substitution list by removing the first substitution so that other substitutions can be applied.

```
crl [sub_per_pt] : < [ TR1 ] , SUBS @ SUBLIST, pt >
     <[substitutingTrace (TR1, SUBS)], getSubstitutionlist(getMessage
      (analyzingTrace(substitutingTrace (TR1, SUBS), nil)),
     elementary(getMessagelist(analyzingTrace(substitutingTrace
                                      (TR1, SUBS), nil))), NIL), pt >
               if not isSatisfiableNF(substitutingTrace (TR1, SUBS)) .
rl [sub_dis] : < [ TR1 ] , SUBS @ SUBLIST, pt >
                                   => < [ TR1 ] , SUBLIST, pt > .
```

Furthermore, an original trace can be naturally transited to a pending trace if it is not a satisfiable normal form, and the pending trace's substitution list is obtained dynamically. It can also be transited to a satisfiable trace if it is a satisfiable normal form (checked by `isSatisfiableNF`). A pending trace can transfer to a satisfiable trace if all rigid messages are unified, and it is thus a satisfiable normal form of some original trace.

```
crl [ot_to_pt] : < [ TR1 ],SUBLIST,ot > => < [ TR1 ],
      getSubstitutionlist(getMessage(analyzingTrace (TR1,nil)),
   elementary(getMessagelist (analyzingTrace(TR1,nil)), NIL), pt >
         if not isSatisfiableNF (TR1) .
crl [ot_to_st] : < [ TR1 ],SUBLIST,ot > => < [ TR1 ],NIL ,st >
                                        if  isSatisfiableNF(TR1) .
crl [pt_to_st] :  < [ TR1 ] , SUBS @ SUBLIST, pt >
                => < [ substitutingTrace(TR1,SUBS) ], NIL, st >
              if isSatisfiableNF SubstitutingTrace(TR1, SUBS)) .
```

The system starts with an initial state:

```
eq init =  < [ Nil ] , NIL , ot > .
```

9.2 Implementation for Authentication

9.2.1 Protocol Description

By our implementation, the protocol description is surprisingly short, which only contains about sixty lines for each protocol. This part mainly has two fragments.

- The first fragment is used to describe the parametric transition relation, which is a kind of rules in the trace generating system in Subsection 9.1.3;
- The other fragment is used to describe a given specification, that is, a security property of the tested protocol.

As defined in the parametric transition relation, each action in a parametric process can be added to the tail of its parametric trace only once. Since each label is distinct in a process, we will check whether an action has been added to its parametric trace by searching its label in the parametric trace. For example, the protocol description for the first flow of multiple Woo-Lam protocol given in Subsection 5.3.3 will be implemented as follows:

```
crl [A_1] : < [ TR1 ], SUBLIST, ot >  =>
             < [ (TR1 . < a(1), o, name(0) >) ], SUBLIST, ot >
                    if not labelinTrace (TR1, a(1)) .
crl [B_1] : < [ TR1 ], SUBLIST, ot >  =>
                     < [ (TR1 . < b(1), i, name(0) > .
                          < b(2), o, name(3) > )], SUBLIST, ot >
                        if not labelinTrace (TR1, b(1)) .
```

In the trace generating system, each satisfiable trace represents a successful run of the protocol. So we will search whether the negation of a specification is reachable in a satisfiable trace. For example, the negation of the authentication property for the Woo-Lam protocol defined in Characterization 5.5 is represented as follows:

```
search [1] in WOOLAMPROTOCOL : init =>*
   < [ TR1 ], NIL, st > such that not (
    labelinTrace(TR1, acc)
     implies (
        ( labelinTrace(TR1, a(3)) and labelbefore(TR1, a(3), acc) and
          equal((getLabelMessage(TR1,acc)), (getLabelMessage(TR1,a(3))))
        )
        or
        ( labelinTrace(TR1, a'(3)) and labelbefore (TR1,a'(3),acc) and
          equal((getLabelMessage(TR1,acc)), (getLabelMessage(TR1,a'(3))))
        )
     )
    ) .
```

The total protocol description of the two-session Woo-Lam protocol is about 40 lines. We will illustrate the whole protocol description for authentication in bounded sessions by the Yahalom protocol in Subsection D.2 of Appendix D as another example.

9.2.2 Other Tested Protocols

We have focused on the authentication property, and performed several tests for some security protocols. Besides the NSPK protocol, the fixed NSPK protocol, the Abadi-Gordon protocol, and the two variations Woo-Lam protocols, we also tested the Yahalom protocol and the Otway-Rees protocol. The detected counterexamples of these two protocols by our implementation are explained as follows.

The Yahalom Protocol

The variation version of the Yahalom protocol we implemented is a simplified one that comes originally from [36]. It is designed to let two principals A and B establish a private session key K, with the help of a server S. The protocol is given flow-by-flow as follows,

$$\begin{aligned} A \longrightarrow B: &\quad A, N_A \\ B \longrightarrow S: &\quad B, N_B, \{A, N_A\}_{K_{BS}} \\ S \longrightarrow A: &\quad N_B, \{B, K, N_A\}_{K_{AS}}, \{A, K, N_B\}_{K_{BS}} \\ A \longrightarrow B: &\quad \{A, K, N_B\}_{K_{BS}}, \{N_B\}_K \end{aligned}$$

We have tested both one-session and two-session of the Yahalom protocol by our tool. The whole two-session protocol description of the Yahalom protocol is in Subsection D.2 of Appendix D. We have found a counterexample during running the two-session protocols, which is shown in Figure 9.1.

Note that the counterexample we detected represents infinitely many attacks. For example, a simple instance of the counterexample introduced above is an attack as follows,

```
Solution 1 (state 535)
states: 536  rewrites: 79350 in 7692722982ms cpu (1039ms real) (0
    rewrites/second)
TR1 --> < b(1),i,name(0),px(10) > . < b(2),o,(name(1),name(11)),{name(0),px(
    10)}k[name(1),name(2)] > . < b'(1),i,(name(0),px(10)),name(11) > . < b'(2),
    o,(name(1),name(21)),{(name(0),px(10)),name(11)}k[name(1),name(2)] > . < b(
    3),i,{(name(0),px(10)),name(11)}k[name(1),name(2)],{name(11)}px(10) > . <
    acc,o,{(name(0),px(10)),name(11)}k[name(1),name(2)],{name(11)}px(10) >
Maude>
```

Figure 9.1: Snapshot of Maude Result for the Yahalom Protocol

$$
\begin{aligned}
I(A) \longrightarrow B: &\quad A, N_I && \text{(a1)}\\
B \longrightarrow I(S): &\quad B, N_B, \{A, N_A\}_{K_{BS}} && \text{(a2)}\\
I(A) \longrightarrow B: &\quad A, N'_I, N_B && \text{(b1)}\\
B \longrightarrow I(S): &\quad B, N'_B, \{A, N'_I, N_B\}_{K_{BS}} && \text{(b2)}\\
I(A) \longrightarrow B: &\quad \{A, N'_I, N_B\}_{K_{BS}}, \{N_B\}_{N'_I} && \text{(a4)}
\end{aligned}
$$

Actually, an intruder can still attack the protocol when N_I and N'_I are substituted to any other possible messages. Thus in the counterexample we detected, they are represented as two variables, $px(10)$ and $px(11)$.

The Otway-Rees Protocol

The variation version of the Otway-Rees protocol we implemented also originally comes from [36]. Similar to the Yahalom protocol, the Otway-Rees protocol is an authentication protocol that generates a fresh session key K through a server S. The protocol is describe informally as follows:

$$
\begin{aligned}
A \longrightarrow B: &\quad N_A, A, B, \{N_A, A, B\}_{K_{AS}}\\
B \longrightarrow S: &\quad N_A, A, B, \{N_A, A, B\}_{K_{AS}}, N_B, \{N_A, A, B\}_{K_{BS}}\\
S \longrightarrow B: &\quad N_A, \{K, N_A\}_{K_{AS}}, \{K, N_B\}_{K_{BS}}\\
B \longrightarrow A: &\quad N_A, \{K, N_A\}_{K_{AS}}
\end{aligned}
$$

We have tested both one-session and two-session of the Otway-Rees protocol by our implementation, and found a counterexample when running the two-session protocol, which is shown in Figure 9.2.

One instance of the counterexample is as follows,

```
Solution 1 (state 2163)
states: 2164  rewrites: 1502171 in 7690700982ms cpu (22316ms real) (0
    rewrites/second)
TR1 --> < a(1),o,((name(10),name(1)),name(2)),{(name(10),name(1)),name(2)}k[
    name(1),name(0)] > . < a'(1),i,((px(20),px(21)),name(1)),px(22) > . < a'(
    2),o,((((px(20),px(21)),name(1)),px(22)),name(31)),{(px(20),px(21)),name(
    1)}k[name(1),name(0)] > . < s(1),i,((((px(20),px(21)),name(1)),{(px(20),px(
    21)),name(1)}k[px(21),name(0)]),name(10)),{(px(20),px(21)),name(1)}k[name(
    1),name(0)] > . < s(2),o,(px(20),{name(8),px(20)}k[px(21),name(0)]),{name(
    8),name(10)}k[name(1),name(0)] > . < a(2),i,name(10),{name(8),name(10)}k[
    name(1),name(0)] > . < acc,o,name(10),{name(8),name(10)}k[name(1),name(0)]
    >
Maude>
```

Figure 9.2: Snapshot of Maude Result for the Otway-Rees Protocol

$$A \longrightarrow I(B): \quad N_A, A, B, \{N_A, A, B\}_{K_{AS}} \tag{a1}$$
$$I \longrightarrow A: \quad N_I, I, A, N'_I \tag{b1}$$
$$A \longrightarrow I(S): \quad N_I, I, A, N'_I, N'_A, \{N_I, I, A\}_{K_{AS}} \tag{b2}$$
$$I(A) \longrightarrow S: \quad N_I, I, A, N'_I, N_A, \{N_I, I, A\}_{K_{AS}} \tag{b2'}$$
$$S \longrightarrow I(A): \quad N_I, \{K, N_I\}_{K_{IS}}, \{K, N_A\}_{K_{AS}} \tag{b3}$$
$$I(B) \longrightarrow A: \quad N_A, \{K, N_A\}_{K_{AS}} \tag{a4}$$

This is a complex attack composed by a man-in-middle attack (in (b2) and (b2')), and a replay attack (in (b3) and (a4)). The places of N_I, N'_I and I can be replaced by any other massages. Thus in the counterexample, they are denoted by three variables, $px(20)$, $px(22)$, and $px(21)$, respectively.

9.2.3 Experimental Results

The results for authentication for security protocols in bounded sessions are summarized in Figures 9.3. In the figure, "Woo-lam protocol" is the two-session Woo-Lam protocols introduced in Subsection 3.3.3, and "Woo-lam protocol*" is a variation described in Subsection 5.5.3.

The number in the column "sessions" is the number of sessions we have modeled when checking the properties. In the column "protocol spec.", the number means the line number for a protocol specific description. Besides that, each Maude file also contains about 330 lines for a common part. The number in the column "states" means the states generated by the system, and the column "times" shows how many milliseconds that checking the protocol takes; In the column "flaws", "detected" means we detected an attack, while"secure" means within the number of sessions, we did not detect any attacks.

There are two possibilities when representing a two-session protocol. One possibility is that, each principal acts in the same role (i.e. a sender or a receiver). This case actually means that a principal initiates two sessions by communicating with an unbounded number of principals, because of the usage of binders. The second possibility is that,

each principal acts in two different roles in the two sessions. This case represents that each of two different principals initiates a session by communicating with an unbounded number of principals, respectively. In both cases, one principal should intend to receive messages from the other in one of two sessions. Otherwise the two principals may have no communications with each other, and thus we could not define any security properties between them. In Figure 9.3, a two-session protocol in which each principal acts in the same role is labeled by †, and in different roles is labeled by ‡.

protocols	session	protocol spec.	states	times(ms)	flaws
NSPK protocol	1	20	46	130	detected
Woo-Lam protocol*	1	25	168	160	detected
fixed NSPK protocol	1	20	164	637	secure
fixed NSPK protocol ‡	2	29	16,468	243,460	secure
Abadi-Gordon protocol	1	20	238	713	secure
Abadi-Gordon protocol †	2	30	4,802	30,499	secure
Yahalom protocol	1	26	279	2,111	secure
Yahalom protocol ‡	2	36	536	1,039	detected
Otway-Rees protocol	1	25	461	8,185	secure
Otway-Rees protocol ‡	2	34	2,164	22,316	detected
Woo-lam protocol	1	25	552	2,460	secure
Woo-lam protocol †	2	51	105,423	476,507	detected

Figure 9.3: Experimental Results for Authentication Protocols

The tests were performed on a Pentium 1.4 GHz, 1.5 G memory PC, under Windows XP. By the experimental results, we could find that a protocol that is secure in one session is not necessarily really secure. For instance, we could not detect any flaws the Yahalom protocol, the Otway-Ree protocol, and the Woo-Lam protocol with one session, while in the two-session of these protocols, flaws do exist.

The checking is quite time-consuming when number of sessions increases. By our experience, checking a protocol with more than three sessions often takes several hours.

9.3 Implementation for Recursive Protocols

The parametric system for recursive protocols shares the same semantics with the parametric system for bounded sessions. The only difference is that there exist infinite transitions in the parametric system for recursive protocols, which are later folded by a pushdown system. Thus the trace generating system for recursive protocols is actually a pushdown system.

9.3.1 Trace Generating System as the Pushdown System

We define the state of the trace generating system for recursive protocols (pushdown system) as a 5-tuple, $\langle ML, tr, STACK, S, k\rangle$, where

- ML is a message list for recursive messages.

- tr is a parametric compaction (a parametric trace with bounded length, in Definition 6.6).
- $STACK$ is a stack with unbounded memory.
- S is a list of substitutions.
- k is a type of tr, where $k \in \{oc, sc, pc\}$. oc denotes an original compaction, sc denotes a satisfiable compaction and pc represents a pending compaction.

In Maude, the state is defined as follows:

```
sort Tracestate Tracetype State .
  ops  oc sc pc : -> Tracetype [ctor] .
  op [_]  : Trace -> Tracestate [frozen] .
  op <_,_,_,_,_> : Messagelist Tracestate Stack
                              Substitutionlist Tracetype -> State .
```

The system starts with an initial state,

```
  eq init =  < nil, [ Nil ], empty , NIL , oc >  .
```

Furthermore, push and pop operations on the stack is defined as follows, in which `star` is the only stack alphabet of the pushdown system.

```
    op push : Stack -> Stack .
    op pop  : Stack -> Stack .

    eq  push (STACK) = star , STACK .

    eq  pop  ( SE, STACK ) = STACK .
    eq  pop  (STACK) = STACK [owise] .
```

The pushdown system has the similar rules to ones defined in Subsection 9.1.3. The only difference is that these rules are defined in a pushdown system.

Firstly, if a parametric compaction is a pending compaction, and its substitution list is not empty, it will be deduced to a new compaction by applying the first substitution in its substitution list. A new substitution list of the new compaction will be calculated dynamically by applying `analyzingTrace` to the new compaction. Secondly, given a state, we also shrink its substitution list by removing the first substitution so that other substitutions can be applied.

```
crl [sub_per_pc] : < ML1, [ TR1 ] , STACK, SUBS @ SUBLIST, pc >
     < ML1, [substitutingTrace (TR1, SUBS)], STACK,
                                    getSubstitutionlist(getMessage
      (analyzingTrace(substitutingTrace (TR1, SUBS), nil)),
     elementary(getMessagelist(analyzingTrace(substitutingTrace
                                        (TR1, SUBS), nil))), NIL), pc >
               if not isSatisfiableNF(substitutingTrace (TR1, SUBS)) .
rl [sub_dis] : < ML1,[ TR1 ] , STACK, SUBS @ SUBLIST, pc >
                            => < ML1, [ TR1 ], STACK, SUBLIST, pc > .
```

Next, an original compaction can naturally transfer to a pending compaction if it is not a satisfiable normal form, and the pending compaction's substitution list is obtained dynamically. It can also transit to a satisfiable compaction if it is a satisfiable normal form (checked by `isSatisfiableNF`). A pending compaction can transit to a satisfiable compaction if all rigid messages are unified.

```
crl [oc_to_pc] : < ML1, [ TR1 ], STACK, SUBLIST, oc > =>
             < ML1, [TR1 ], STACK, getSubstitutionlist
        (getMessage(analyzingTrace (TR1,nil)),
   elementary(getMessagelist (analyzingTrace(TR1,nil)), NIL), pc >
                   if not isSatisfiableNF (TR1) .
crl [oc_to_sc] : < ML1, [ TR1 ], STACK, SUBLIST, oc > =>
         < ML1, [ TR1 ], STACK, NIL ,sc>
                              if  isSatisfiableNF(TR1) .
crl [pc_to_sc] :  < ML1, [ TR1 ], STACK, SUBS @ SUBLIST, pc >
            => < [ substitutingTrace(TR1,SUBS) ], NIL, sc >
          if isSatisfiableNF SubstitutingTrace(TR1, SUBS)) .
```

9.3.2 Protocol Description

The main difference between the pushdown system and previous trace generating system is that, for the recursive process, one should apply push, when enters a recursive process, and apply pop, when returns from a recursive process. For example, in the description of the recursive authentication protocol (in Subsection 3.4.2), push and pop actions are illustrated as follows:

```
crl [B_1] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
              < ML1, [ ( TR1 .
                < b(1), i, H(lk[MA,name(1)],
                              (((MA,A[MA]),px(21)),px(22)) )  > .
                < b(2), o, H(lk[A[MA],name(1)],
                  (((A[MA],A[A[MA]]),N[px(21)]),H(lk[MA,name(1)],
                              (((MA,A[MA]),px(21)),px(22)) )) ) >
                       ) ], push (STACK),  SUBLIST, oc >
                              if noc labelinTrace (TR1, b(1)) .
crl [B_3] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
            < ML1, [ ( TR1 . < b(1), i, H(lk[MA,name(1)],
                               (((MA,A[MA]),px(21)),px(22)) )  > .
               < b(3), o, H(lk[A[MA],name(1)],
                 (((A[MA],name(1)),N[px(21)]),H(lk[MA,name(1)],
                        (((MA,A[MA]),px(21)),px(22)))) ) >
                        ) ], pop (STACK) ,  SUBLIST, oc >
```

The total protocol description of the RA protocol is in Subsection D.3 of Appendix D.

A counterexample for the authentication specification (see Characterization 6.1) is detected automatically. The result snapshot is in Figure 9.4.

in which `MA`, `MN`, and `Mk` are binder markers defined in Definition 6.5. `name(1)` denotes the server name S. This counterexample is the exactly one we introduced in Subsection 6.4.

```
Solution 1 (state 415)
states: 416  rewrites: 67367 in 7689676981ms cpu (824ms real) (0
    rewrites/second)
ML1 --> (((k[Mk],name(1)),px(32))lk[px(31),name(1)],((Mk,px(33)),px(32))lk[px(
    31),name(1)]),((Mk,px(31)),px(34))lk[px(33),name(1)]
TR1 --> < a(1),o,H(lk[MA,name(1)],((MA,A[MA]),MN),null) > . < b(1),i,H(lk[MA,
    name(1)],((MA,A[MA]),MN),null) > . < b(3),o,H(lk[A[MA],name(1)],((A[MA],
    name(1)),N[MN]),H(lk[MA,name(1)],((MA,A[MA]),MN),null)) > . < s(1),i,H(lk[
    A[MA],name(1)],((A[MA],name(1)),N[MN]),H(lk[MA,name(1)],((MA,A[MA]),MN),
    null)) > . < s(2),o,(((k[Mk],name(1)),N[MN])lk[A[MA],name(1)],((Mk,MA),N[
    MN])lk[A[MA],name(1)]),((Mk,A[MA]),MN)lk[MA,name(1)] > . < a(2),i,((Mk,A[
    MA]),MN)lk[MA,name(1)] > . < acc,o,((Mk,A[MA]),MN)lk[MA,name(1)] >
STACK --> empty
```

Figure 9.4: Snapshot of Maude Result for the Recursive Authentication Protocol

In comparison, we also tested the fixed RA protocol defined in Subsection 6.4, and did not find its counterexamples.

9.3.3 Experimental Results

The results of two tests are summarized in Figure 9.5. Note that all experiments are tested with one session bound. In the figure, column "protocol spec." is the number of lines for a protocol description. In addition to these lines, each Maude file also contains about 400 lines for the common description of the method.

protocols	protocol spec.	states	times(ms)	flaws
recursive authentication protocol	32	416	824	detected
fixed recursive authentication protocol	32	416	1,068	secure

Figure 9.5: Experimental Results for Recursive Protocols

Remark 9.3. It is exact the last state generated by the parametric system for the RA protocol that has flawed, so the numbers of states for the RA protocol, and for the fixed RA protocol are surprisedly same.

9.4 Implementation for Non-repudiation and Fairness

9.4.1 Protocol Description

As we have assumed, a dishonest principal can abort the protocol run at any stage, or continue to communicate with the other principal. In the implementation, we use multiple rewrite rules. For example, according to the second and third flows in the simplified ZG protocol (refer to Subsection 7.1.3), when A receives a message from B, it aborts the protocol by stoping sending the message (described by the [A_2] rule as follows), or continue running the next flow (described by the following [A′_2] rule).

```
crl [A_2] : < [ TR1 ], SUBLIST, ot >  =>
        < [ (  TR1 .  < label(name(0),2), i, {((name(0),name(11)),
            px(12))}-k[name(1)] > .  STOP(name(0)) ) ], SUBLIST, ot >
        if  labelinTrace (TR1, label(name(0),1)) and
                              not labelinTrace (TR1, label(name(0),2)) .
crl [A'_2] : < [ TR1 ], SUBLIST, ot >  =>
      < [ TR1 .  < label(name(0),2), i, {((name(0),name(11)),px(12))}
     -k[name(1)] > . < label(name(0),3), o, {((px(17),px(18)),px(19))}
                                          -k[name(0)] > ], SUBLIST, ot >
        if  labelinTrace (TR1, label(name(0),1)) and
                             not labelinTrace (TR1, label(name(0),2)) .
```

The total protocol description of the simplified ZG protocol for FAIRO is in Subsection D.4 of Appendix D.

9.4.2 Other Tested Protocols

In experiments with one session bound, the attacks for NRO, FAIRO and FAIRM of the simplified ZG protocol were detected automatically. For comparison, we also implemented the analysis for the full ZG protocol [116], which guarantees those three properties. Furthermore, we also tested some protocols proposed by the International Organization for Standardization [64, 65, 66]. It is well-known that these protocols have flaws [73, 115] (although no other formal verifications are applied to these examples).

Figure 9.6 shows the snapshot of the result of NRO for simplified ZG protocol. In the counterexample, `name(0)`, `name(1)`, `name(2)`, `name(11)` denote names A, B, S, N_A, respectively, and `px(31)` denotes a variable that can substitute to any messages A generates.

```
Solution 1 (state 512)
states: 513  rewrites: 316892 in 7694712980ms cpu (3954ms real) (0
    rewrites/second)
TR1 --> < label(name(0), 1),o,{(name(1),name(11)),px(33)}-k[name(0)] > . <
    label(name(1), 1),i,{(name(1),name(11)),px(33)}-k[name(0)] > . < label(
    name(1), 2),o,{(name(0),px(25)),px(26)}-k[name(1)] > . < label(name(2), 1),
    i,{(name(1),name(11)),px(33)}-k[name(0)] > . < label(name(2), 2),o,{((name(
    0),name(1)),name(11)),px(33)}-k[name(2)] > . < label(name(2), 3),o,{((name(
    0),name(1)),name(11)),px(33)}-k[name(2)] > . < label(name(1), 3),i,{((name(
    0),name(1)),name(11)),px(33)}-k[name(2)] > . < evida,o,{(name(1),name(11)),
    px(33)}-k[name(0)],{((name(0),name(1)),name(11)),px(33)}-k[name(2)] >
```

Figure 9.6: Snapshot of Maude Result for NRO of the Simplified ZG protocol

The counterexample actual represents the same attack introduced in Subsection 7.4. Similarly, a counterexample for FAIRM is also detected automatically. The result snapshot is in Figure 9.7.

The ISO/IEC13888-2 M2 Protocol

The ISO/IEC 13888-2 [65] contains three non-repudiation protocols (M1, M2, and M3), providing non-repudiation of origin and non-repudiation of receipt service using

```
Solution 1 (state 4108)
states: 4109  rewrites: 3491942 in 7689004980ms cpu (45545ms real) (0
    rewrites/second)
TR1 --> < label(name(0), 1),o,{(name(1),name(11)),px(33)}-k[name(0)] > . <
    label(name(1), 1),i,{(name(1),name(11)),px(33)}-k[name(0)] > . < label(
    name(1), 2),o,{(name(0),name(11)),px(33)}-k[name(1)] > . < label(name(0),
    2),i,{(name(0),name(11)),px(33)}-k[name(1)] > . < label(name(0), 3),o,{(px(
    17),px(18)),px(33)}-k[name(0)] > . < label(name(2), 1),i,{(name(1),name(
    11)),px(33)}-k[name(0)] > . < label(name(2), 2),o,{((name(0),name(1)),name(
    11)),px(33)}-k[name(2)] > . < label(name(2), 3),o,{((name(0),name(1)),name(
    11)),px(33)}-k[name(2)] > . < check,i,px(18) > . < label(name(0), 4),i,{((
    name(0),name(1)),name(11)),px(33)}-k[name(2)] > . < evidb,o,{(name(0),name(
    11)),px(33)}-k[name(1)],{((name(0),name(1)),name(11)),px(33)}-k[name(2)] >
```

Figure 9.7: Snapshot of Maude Result for FAIRM of the Simplified ZG protocol

symmetric techniques. We have tested the M2 protocol, and detected the attacks for NRO and FAIRO automatically. The M2 Protocol is given as follows:

$$A \longrightarrow S: \quad \{F_{NRO}, A, B, \mathcal{H}(M)\}_{K_{AS}} \tag{M2-1}$$
$$S \longrightarrow A: \quad \{\{F_{NRO}, A, B, \mathcal{H}(M)\}_{-K_S}\}_{K_{AS}} \tag{M2-2}$$
$$A \longrightarrow B: \quad M, \{F_{NRO}, A, B, \mathcal{H}(M)\}_{-K_S} \tag{M2-3}$$
$$B \longrightarrow S: \quad \{\{F_{NRO}, A, B, \mathcal{H}(M)\}_{-K_S}\}_{K_{BS}} \tag{M2-4}$$
$$S \longrightarrow B: \quad \{\{F_{NRO}, A, B, \mathcal{H}(M)\}_{-K_S}, \{F_{NRR}, A, B, \mathcal{H}(M)\}_{-K_S}\}_{K_{BS}} \tag{M2-5}$$
$$S \longrightarrow A: \quad \{\{F_{NRR}, A, B, \mathcal{H}(M)\}_{-K_S}\}_{K_{BS}} \tag{M2-6}$$

in which the evidence of origin is $\{F_{NRO}, A, B, \mathcal{H}(M)\}_{-K_S}$, and the evidence of receipt is $\{F_{NRR}, A, B, \mathcal{H}(M)\}_{-K_S}$.

In the M2 protocol, NRO is established in the first five flows. At flow (M2-1), the sender A asks the TTP server S to generate evidence of origin for M to be transferred to the receiver B. Other principals cannot impersonate A to make such a request, as A's request is protected by the symmetric key K_{AS} shared with A and S. After obtaining evidence of origin from S at flow (M2-2), A sends M and evidence of origin to B at flow (M2-3). B should validate M before proceeding to the flow (M2-4). Only when this validation succeeds will B send evidence of origin to S for verification and request evidence or receipt from the server at the same time. After that, S will directly send the evidence of receipt to A at flow (M2-6).

In the M2 protocol, once B gets the confirmed evidence of origin from the server, A will be guaranteed to be provided with evidence of receipt by the server. However, when A sends M and evidence of origin to B at flow (M2-3), it always leaves B in an advantageous position. Upon receiving M, B may first validate the M before proceeding to next flow. If B aborts the protocol at flow (M2-4), no one will have evidence that B has received M. This attack violates both NRO and FAIRO, which is detected automatically in our experiments.

The ISO/IEC13888-3 M-h Protocol

ISO/IEC 13888-3 [66] specifies two separate protocols (M-h, M-e) for non-repudiation service without the involvement of online and inline TTPs. We used the M-h protocol as a case study, and detected the attacks for FAIRO automatically. The M-h Protocol is given flow-by-flow as follows:

$$A \longrightarrow B: \quad F_{POE}, B, \mathcal{H}(M), \{F_{POE}, B, \mathcal{H}(M)\}_{-K_A} \quad \text{(M-h-1)}$$
$$B \longrightarrow A: \quad F_{ACK}, A, \{F_{ACK}, A, \mathcal{H}(M)\}_{-K_B} \quad \text{(M-h-2)}$$
$$A \longrightarrow B: \quad F_{NRO}, B, M, \{F_{NRO}, B, M\}_{-K_A} \quad \text{(M-h-3)}$$
$$B \longrightarrow A: \quad F_{NRR}, A, \{F_{NRR}, A, M\}_{-K_B} \quad \text{(M-h-4)}$$

in which evidence of origin is $\{F_{NRO}, B, M\}_{-K_A}$, evidence of receipt is $\{F_{NRR}, A, M\}_{-K_B}$.

The M-h protocol uses a one-way hash function to construct the *promise of exchange* and the *acknowledgment* at flow (M-h-1) and (M-h-2). A first sends a promise of exchange that does not reveal the contents of the message. After receiving an acknowledgment from B, A then sends M with evidence of origin and waits for evidence of receipt from B. Actually, this does not solve the non-repudiation problem, since B can still refuse to send the last message, leaving A without evidence of receipt. This does not satisfy the FAIRO property. Our implementation can capture this attack automatically.

9.4.3 Experimental Results

The results of all tests are summarized in Figure 9.8. Note that all experiments are with one session bound. In the figure, column "protocol spec." is the number of lines for a protocol specific description. In addition to these lines, each Maude file also contains about 400 lines for the common description of the method.

protocols	property	protocol spec.	states	times(ms)	flaws
Simplified ZG protocol	NRO	50	513	3,954	detected
	NRR	50	780	3,905	secure
	FAIRO	55	770	2,961	detected
	FAIRR	55	846	3,903	secure
	FAIRM	50	4,109	45,545	detected
Full ZG protocol	NRO	50	633	7,399	secure
	FAIRO	55	788	3,394	secure
	FAIRM	60	788	3,490	secure
ISO/IEC13888-2 M2	NRO	50	1,350	7,710	detected
	FAIRO	65	1,977	6,827	detected
	FAIRR	65	2,131	7,506	secure
ISO/IEC13888-3 M-h	FAIRO	60	295	918	detected
	FAIRR	60	305	1,040	secure

Figure 9.8: Experimental Results for Fair Non-repudiation Protocols

Chapter 10

Related Work

As we know, usually the use of formal methods can be divided into three steps.

Modeling. The first and most delicate phase of formal methods is to convert the system we want to verify into a mathematical model. This generally requires us to abstract away unimportant details, in order to achieve a concise model which is sufficiently simple to be verified.

Specification. The specification is about to state which properties must hold on the given model. For instance, the set of action terms in our model. Temporal logics are common formalisms for describing the properties.

Verification. Finally, one must prove that a model satisfies a given specification. The first way is to verify it manually. However, this approach is obviously problematical and fallible. In recent years, computer-aided verification have become more mature, many technique are proposed, including *model checking*, a fully automatic technique; *theorem proving*, a proof technique relies on human intervention; *resolution*, a technique between model checking and theorem proving. It can be completely automatic, but sometime it will not terminate.

This chapter firstly introduces various well-known models for describe security protocols and their specifications. These models are all applied by more than one verification techniques. Then many applications based on three most well-known verification techniques, model checking, resolution and higher-order theorem proving are introduced respectively.

10.1 Modeling and Specification

10.1.1 Belief Logics

Adopting *belief logics* for security protocols experienced great popularity at the end of the 1980s and beginning of the 1990s. The earliest one was the BAN logic [36], one of the first attempts to make the reasoning about the properties of security protocols more systematic. After that, many logics were proposed, extended from BAN logic (so called *BAN-family* logics), such as GNY logic [54], AT logic [10] SVO logic [110].

The BAN logic is a logic for authentication property. It explicitly assumes that all principals are honest, and thus reasons about the beliefs of these principals. Examples of the formula are:

- $P|\equiv X$: The principal P believes X.
- $P \lhd X$: The principal P can see X, i.e. received a message that contains X.
- $P|\sim X$: The principal P sent a message that contains X.
- $P \overset{K}{\leftrightarrow} Q$ P and P share the symmetric key K.
- $E_K(X)$: the encryption of X under key K.
- $\sharp(X)$: X is fresh.
- etc.

A set of inference rules is proposed, which is used to derive the goals a protocol achieves. For instance,

$$\frac{P|\equiv\ P \overset{K}{\leftrightarrow} Q \quad P \lhd E_K(X)}{P|\equiv\ Q|\sim X}$$

means that if P believes that it and Q share the K, and receives an encryption message encrypted by K, then P believes that the plain message of the encryption message comes from Q.

$$\frac{P|\equiv\ \sharp(X) \quad P|\equiv\ Q|\sim X}{P|\equiv\ Q|\equiv X}$$

means that if P believes that Q sent X, which P believes it is fresh, then P believes that Q believes X (otherwise Q would not sent it).

In 1990, Nessett demonstrated that a significant flaw exists in the BAN logic. He proposed a protocol named the Nessett protocol with flaws that cannot be analyzed by the BAN logic [88]. Thus, the GNY logic was proposed to fix this flaw. The major contributions of the GNY logic was that the notion of *possession*. As a consequence, the Nessett protocol can be proved by GNY logic, such that an intruder can possess the exchanged key [54].

The AT logic was closer to traditional modal logics than the BAN logic. It provided a detailed model of computation. It was more expressive, and had a soundness result with respect to the model.

The SVO logic was probably the most mature one of the belief logics. It was proposed with intension to unify all belief logics for security protocols. It was not just simply to patch on new notations and rules adequately expressive to capture the additional scope of these logics. Rather to produce a model of computation and a logic that is sound with respect to that model while still retaining the expressiveness of all BAN-family logics.

For a detailed survey of logics for security protocols, please refer to [109].

To the best of my knowledge, most verifications based on BAN-family logics are done manually. For example, Jianying Zhou, one of the core researchers in non-repudiation protocols, had proposed several non-repudiation protocols, and verified their correctness

by SVO logic in his papers and book [116, 115, 117]. An automated support result was that the BGNY logic designed by Brackin, based on the GNY logic. He developed an associated automated HOL tool, AAPA (Automated Authentication Protocol Analyzer) [33].

10.1.2 CSP

The language of *Communicating Sequential Processes* (CSP) [61] is a process algebra providing a well-defined semantics and a precise way for reasoning about concurrent systems in general. It is also one of the first attempts to analyze the properties of security protocols.

Processes in CSP are described by the *events* that they will engage in. There are many versions of CSP. The following definition is one variation for security protocol analysis [79].

Definition 10.1 (Processes)**.** The set of processes is defined by the grammar,

$$
\begin{array}{ll}
P ::= & \\
\quad stop & \text{Nil} \\
\quad c!i.j.v \rightarrow P & \text{output} \\
\quad c?i.j.x \rightarrow P & \text{input} \\
\quad P \Box Q & \text{choice} \\
\quad P|[E]|Q & \text{composition}
\end{array}
$$

Intuitively,

- The process *stop* does not engage in any event.
- The process $c!i.j.v \rightarrow P$ sends v on channel c, with source i and destination j. After this event, it behaves as P.
- The process $c?i.j.x \rightarrow P$ awaiting an input on channel c with source i and destination j, after which, it behaves as $P\{x\}$, i.e., x is instantiated.
- $P \Box Q$ behaves either as P or Q.
- $P|[E]|Q$ is the composition of P and Q that have to synchronize on any event in the set of events E. $P|||Q$ is a shortcut for $P|[\emptyset]|Q$ and $|||_i P_i$ is the indexed form.

Using CSP processes, a general model for security protocols has the following form:

$$NET = (|||_{(i \in USER)} USER_i)|[trans, rec]|INTRUDER$$

The intruder process $INTRUDER$ is described by a recursive process with complex structures [79].

For analyzing security protocol, CSP has two kinds of semantics, trace semantics and failure semantics. The former is used to analyze secrecy and authentication properties [79, 102]; The latter is used to analyze non-repudiation and fairness properties [103].

There are two main approaches based on the idea of modeling security protocols in CSP, model checking, and theorem proving.

- The model checking approach used the FDR (Failures Divergences Refinement Checker), a finite model checker, to verify protocols [79]. A specialized compiler, Casper, can translate a protocol description to its corresponding CSP process [83].

- The theorem proving approach was mainly presented by S. Schneider et al.. In [102], he introduced the idea of *rank functions*. He implemented the approach by PVS [52], and applied it to lots of security properties [52, 34, 104, 103].

10.1.3 Spi and Spi-like Calculi

Martín Abadi and Andrew Gordon developed the *Spi calculus* [7] with primitives representing the cryptographic operations of encryption and decryption. In 2002, Martín Abadi and Cédric Fournet proposed a more general process calculus for security protocol analysis, named the *applied pi calculus* [6]. After that, numerous analysis and verifications for security properties are proposed based on these two calculi and their extensions [7, 8, 31, 24, 28, 4, 3, 55, 56, 22, 25, 23, 5].

We simply introduce syntax and semantics of Spi calculus here.

Definition 10.2 (Terms)**.** The set of terms is defined by the grammar,

$L, M, N ::=$		
	n	name
	x	variable
	(M, N)	pair
	0	zero
	$suc(M)$	successor
	$\{M\}_K$	encryption

Definition 10.3 (Processes)**.** The set of processes is defined by the grammar,

$P, Q, R ::=$		
	$\mathbf{0}$	nil
	$\overline{M}\langle N\rangle.P$	output
	$M(x).P$	input
	$P \mid Q$	composition
	$(\nu n)P$	restriction
	$!P$	replication
	$[M \ is \ N]P$	match
	$let \ (x, y) = M \ in \ P$	pair splitting
	$case \ M \ of \ 0 : P \ \ suc(x) : Q$	integer case
	$case \ L \ of \ \{x\}_K \ in \ P$	key decryption

The semantics of Spi calculus is defined by the following three relations, reduction relation, structural equivalence and reaction relation.

Definition 10.4 (reduction relation)**.** The reduction relation on closed processes is defined by the following rules,

$$
\begin{aligned}
!P &> P \mid !P \\
[M \ is \ M]P &> P \\
let \ (x, y) = (M, N) \ in \ P &> P[M/x][N/y] \\
case \ 0 \ of \ 0 : P \ \ suc(x) : Q &> P \\
case \ suc(M) \ of \ 0 : P \ \ suc(x) : Q &> Q[M/x] \\
case \ \{M\}_K \ of \ \{x\}_K \ in \ P &> P[M/x]
\end{aligned}
$$

Definition 10.5 (structural equivalence)**.** The structural equivalence is the least relation on closed processes that satisfies the following equations and rules,

$$\begin{array}{rcl} P|\mathbf{0} & \equiv & P \\ P|Q & \equiv & Q|P \\ P|(Q|R) & \equiv & (P|Q)|R \\ (\nu m)(\nu n)P & \equiv & (\nu n)(\nu m)P \\ (\nu n)\mathbf{0} & \equiv & \mathbf{0} \\ (\nu n)(P|Q) & \equiv & P|(\nu n)Q \quad if\ n \notin f_n(P) \end{array}$$

$$\frac{P > Q}{P \equiv Q} \qquad \frac{}{P \equiv P} \qquad \frac{P \equiv Q}{Q \equiv P}$$

$$\frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \qquad \frac{P \equiv Q}{P|R \equiv Q|R} \qquad \frac{P \equiv Q}{(\nu n)P \equiv (\nu n)Q}$$

Definition 10.6 (reaction relation)**.** Let the reaction relation be the least relation on closed processes that satisfies,

$$\overline{M}\langle N\rangle.P|M(x).Q \rightarrow P|Q[N/x]$$

$$\frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q} \qquad \frac{P \rightarrow Q}{P|R \rightarrow Q|R} \qquad \frac{P \rightarrow Q}{(\nu n)P \rightarrow (\nu n)Q}$$

The verification approaches based on Spi and Spi-like calculi almost cover all formal methodologies.

- The first verification approach for Spi and Spi-like calculi was done using equivalence [7, 8, 31, 24]. In this approach, a security protocol and its specification for a security property were described by two processes, respectively. If the two processes satisfied some equivalence, for instance, test equivalence [7], then the security protocol satisfied the given security properties. Unfortunately, these equivalences were usually undecidable. Recently, some automatic tools were developed, enforcing restrictions on these calculi [49, 24]. For instance, removing the replication operators, enforcing an upper-bound on the depths of reduction and reaction, etc..

- Another approach based on these process calculi was a static analysis based on type systems [4, 3, 55, 56]. The limitation of this method was that the intruder's model is weaker than the Dolev-Yao model, assuming that the intruder was partially trusted. An example for secrecy was to assign each original message one of three types `PUBLIC`, `PRIVATE`, and `ANY`, then to infer whether a confidential message keeps type `PRIVATE` during transitions.

- M. Abadi and B. Blanchet translated a process of an applied calculus to a set of Horn clauses [5]. Thus security properties can by analyzed by resolution [22, 23, 25, 5]. Abadi and Blanchet had also established an equivalence between typed process calculus technique and logic program technique with respect to secrecy property [5].

- Michele Boreale proposed a model checking method on a variation of Spi calculus [28] to analysis secrecy and authentication properties for bounded sessions security protocols. He represented intruders as a deductive system, and thus replication/recursion operation were avoided without weakening the ability of intruders. Our approach is based on his work.

10.1.4 Strand Space

The *strand space* formalism [58, 59] is a framework for studying security protocol analysis developed by Thayer, Herzog and Guttman. Intuitively speaking, a *strand* represents the sequence of actions in which a particular protocol principal may participate. For example, a sender strand for the NSPK protocol is,

$$\langle +\{N_A, A\}_{+K_B}, -\{N_A, N_B\}_{+K_A}, +\{N_B\}_{+K_B} \rangle$$

The + and − signs signify whether the message is sent or received, respectively. We refer to each element of the strand as a *node*. Similarly, a receiver strand for the NSPK protocol is,

$$\langle -\{N_A, A\}_{+K_B}, +\{N_A, N_B\}_{+K_A}, -\{N_B\}_{+K_B} \rangle$$

A number of strands represent the intruder's possible interactions. These strands can be adjusted in we want to tailor the capabilities of intruders.

Strands (include intruders') may interact according to the ways in which nodes can exchange messages. Thus a message sent by one principal may correspond to that received by another principal.

Two kinds of edge are introduced to the strands, $\Longrightarrow$ and $\longrightarrow$. $\Longrightarrow$ connects successive nodes of a strand. $\longrightarrow$ indicates the causal ordering that arises from the transmission and subsequent reception of a message. Analysis in a strand space is carried out on a particular structure: *bundles*. For example, a bundle showing the attack on the NSPK protocol is in Figure 10.1.

A bundle must be well-founded. That is, whenever it contains a reception node $-n$, then it also contains a unique transmission node $+n$. On the other hand, a transmission node might correspond to many reception nodes or none. Bundles thus take on the structure of acyclic, ordered graphs.

The strand space method had been implemented in an automated tool Athena [107]. The tool took as input a slightly modified version of the strand space model, that are parametric strands. This means that the strands may contain variables associated with the roles. In [107], Song proposed a model-checking algorithm that may not terminate if no bound is given on the number of sessions. Furthermore, lots of other formal methods were applied to strand space [2], which was still subject to active research.

10.1.5 HLPSL

The *High Level Protocol Specification Language* (HLPSL) [15] is an expressive, modular, role-based, formal language that allows for the specification of control flow patterns, data-structures, alternative intruder models, complex security properties, as well as different cryptographic primitives and their algebraic properties. These features make HLPSL

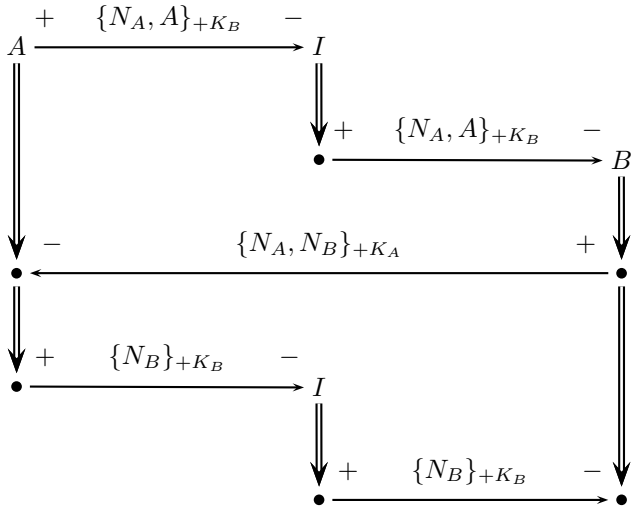


Figure 10.1: The Strand Space Bundle for the NSPK Protocol

well suited for specifying modern, industrial-scale protocols. Moreover, the HLPSL enjoys both a declarative semantics based on a fragment of the *Temporal Logic of Actions* and an operational semantics based on a translation into a rewrite-based formalism, *Intermediate Format* (IF).

IF is also a tool-independent protocol specification language suitable for automated deduction. The main goal in the design of the IF was to provide a low-level description of the protocol that is suitable for automatic analysis (rather than being abstract and easy to read for human users like the HLPSL), and yet this format should be independent from the analysis methods employed by the various back-ends.

Subfigure (a) in Figure 10.2 shows the HLPSL specification of the Yahalom protocol, and its corresponding initial states in IF is shown in Figure 10.2, (b) (which is automatically generated by HLPSL2IF translator).

HLPSL is the language through which end users make use of the AVISPA (Automated Validation of Internet Security Protocols and Applications) tool-set [1], which employs four tools to tackle validation of security protocols.

The On-the-fly Model-Checker (OFMC) [17] performed protocol falsification and bounded verification by exploring the transition system described by an IF specification in a demand-driven way. OFMC implemented a number of correct and complete symbolic techniques. It supported the specification of algebraic properties of cryptographic operators, and typed and untyped protocol models.

The Constraint-Logic-based Attack Searcher (CL-AtSe) [16] applied constraint solving with some powerful simplification heuristics and redundancy elimination techniques. CL-AtSe was built in a modular way, and was open to extensions for handling algebraic properties of cryptographic operators. It supported type-flaw detection and handles associativity of message concatenation.

The SAT-based Model-Checker (SATMC) [42] built a propositional formula en-

```
Protocol Yahalom;
  Identifiers
    A,B,S: role;
    k: function;
    KAB: symmetric_key;
    NA,NB: nonce;
  Knowledge
    A: B,S,k(A,S);
    B: A,S,k(B,S);
    S: A,B,k;
  Messages
    1. A -> B: A,NA
    2. B -> S: B,{|A,NA,NB|}k(B,S)
    3. S -> A: {|B,KAB,NA,NB|}k(A,S),{|A,KAB|}k(B,S)
    4. A -> B: {|A,KAB|}k(B,S),{|NB|}KAB
  Session_instances
    [A:a; B:b; S:s]
    [A:i; B:b; S:s];
  Intruder_knowledge A,B,S;
  Goal B authenticates S on KAB
```

(a) HLPSL Specification

```
state(roleA,step0,sess1,a,b,s,k(a,s)).
state(roleB,step0,sess1,a,b,s,k(b,s)).
state(roleS,step0,sess1,a,b,s,k).
state(roleB,step0,sess2,i,b,s,k(b,s)).
state(roleS,step0,sess2,i,b,s,k).
i_knows(a).i_knows(b).i_knows(s).
i_knows(i).i_knows(k(i,s))
```

(b) IF Specification

Figure 10.2: HLPSL and IF Specifications of the Yahalom Protocol

coding a bounded unrolling of the transition relation specified by the IF, the initial state and the set of states representing a violation of the security properties. The propositional formula was then fed to a state-of-the-art SAT solver and any model found was translated back into an attack.

The Tree Automata based on Automatic Approximations(TA4SP) [26] approximated the intruder knowledge by using regular tree languages and rewriting. For secrecy properties, TA4SP could show whether a protocol is flawed (by under approximation) or whether it is safe for any number of sessions (by over approximation).

10.1.6 Other Formalisms

Besides the formalisms introduced above, various other formalisms for the specification of security protocols had been proposed. For instance, higher-order logic by Paulson et al. [95, 18], first-order logic by C. Weidenbach [112], Linear logic by N. Durgin et al. [51], equational logic by J. Jacquemard et al. [67], hidden algebra by K. Ogata et al. [90], and tree automata by several different groups [86, 75, 26].

10.2 Validation

10.2.1 Model Checking

Model checking is an algorithmic verification technique by exhaustively searching the state space of the underlined model. It is attractive for enjoying two advantages: (1) it is fully automatic; (2) counterexamples are provided once a model checking fails.

There are not many well-known researches applying traditional model check methods such as finite system (Kripke) model checking [39], or automata theoretical based model checking [105], due to two facts. Firstly, it is extremely difficult to describe a security protocol model if one considers all infinity factors. Secondly, specifications for security properties are rather simple. Almost all well-known properties can be specified as a safety problem. To the best of my knowledge, many model checking of security protocols adopted a rich model, such as process algebra, to describe security protocols, and proposed non-traditional ways to check the security properties. The following is an inexhaustive survey.

Gavin Lowe first used process calculus CSP to describe behaviors of security protocols, and implemented a model-checker FDR to discover numerous attacks of various security protocols [80, 81]. In his work, the intruder is represented as a recursive process. He restricts the state space to be finite by imposing upper-bounds upon messages the intruder generates, and also upon the principals in the network. Thus security properties can be checked on a finite trace models.

Many of our ideas are inspired by the symbolic approach of Michele Boreale et al. [28]. They used a variation of Spi calculus, and proposed a trace semantics for the calculus. An environmental deductive system is proposed to represent abilities of intruders, and thus replication/recursion operations can be avoided without weakening the ability of intruders. They implemented their methodology by a tool named STA [29], restricting the number of principals and intruders, and enforcing each principal explicitly communicates with an intruder. Our model finitely represents an unlimited number of principals and intruders in the network with binders. This is more powerful than his.

David Basin et al. proposed an On-the-fly model checking method (OFMC) [17]. They used a high-level language, HLPSL, to represent a protocol, which then translates automatically to a low-level language, IF. An intruder's messages were instantiated when necessary, which was similar to the occasion when a rigid message occurs in our model. In their work, an intruder's role was explicitly assigned, for instance, as a sender. This was efficient, but the process needed to be performed several times to ensure that no intruders can attack a protocol in any roles. In our work, we do not explicitly define an intruder, and we have to check all situations in which intruders act in different roles at one time.

Song chose the strand space to represent security protocols, and proposed a model-checking algorithm to check security properties on strand spaces, and implemented a model checker named Athena [107]. The algorithm, although efficient, might not terminate if no bounds were given on the number of sessions. However, he showed that for most practical protocols, it would terminate even without bound of sessions.

There were several studies based on game-theoretic model checking method on the fairness property. S. Kremer firstly analyzed various these protocols, and also summarized and compared many formal definitions of fairness in his thesis [72]. Recently, D. Kähler et al. proposed a more powerful AMC-model checking method for verifying the fairness

property [68].

10.2.2 Resolutions

Recently, security protocol analysis based on resolution has been flourishing, and numerous researches have been proposed.

In a security protocol, each action of a principal running a protocol can be regarded as a reaction when it receives a message. Thus it is naturally described as a horn clause. By this means, a security protocol can be translated to a finite set of Horn clauses, and resolution can be applied on them.

This approach can verify unbounded sessions of a protocol by over-approximations, when each horn clause is permitted to be repeatedly applied. Although theoretically the termination of a resolution is not guaranteed, practically it often efficiently verifies the target.

The method is naturally adopted for analyzing secrecy property. Authentication property can be translated to a secrecy property, so that it can also be handled by this approach [23]. However, Due to difficulties to represent dishonest principals, It seems that other security properties, such as non-repudiation and fairness, are difficult to be analyzed. The other problem is to extract a counterexample when finding a flaw. Some researches are towards this approach [12].

M. Abadi and B. Blanchet first translated a process of an applied calculus to a set of Horn clauses [5]. Thus security protocols analysis can be discussed in the resolution methodology [22, 23, 25, 5, 12]. It verified the secrecy property in infinite sessions of a protocol with infinite principals by some approximations on both sessions and principals [22, 25]. The method sometimes did not terminate, as the author noted. A tag system that assigned each encrypted message a unique tag was added to the system to make it terminate [25]. The authors proved that security of a tagged protocol did not imply the security of an untagged version. Furthermore, B. Blanchet proposed a transformation between two processes, and proved that the former process satisfying authentication property is equal to the latter one satisfying secrecy property [23]. Thus authentication can also be analyzed within this formalism.

T. Truderung used resolution to analyze secrecy property for recursive protocols by proposing a selecting theory [111]. Recently, R. Küsters and T. Truderung considered the arithmetic algorithm for the recursive protocol by generalizing the previous model [74], detecting the known attack [98] of the RA protocol automatically. Our model does not consider which encryption algorithm protocols adopted, and thus we cannot find the attack.

The research of Comon-Lundh and Cortier [41] was also based on the Horn clauses. They proved that it is sufficient to consider only a bounded number of principals when verifying secrecy and authentication properties. They distinguished intruders as compromised principals and eavesdroppers, reducing a system with infinite principals to one with finite principals.

10.2.3 Theorem Proving

Higher-order theorem proving is an inductive approach capable of verifying infinite state space. It depends heavily on human's guidance, and thus usually needs expertise

and heuristics.

Although theorem proving approach can handle infinite state space, Existing researches for security protocol analysis still chose to enforce restrictions on infinity factors for security protocols, making it easily be proved.

There are two groups mainly contribute to this approach. One is S. Schneider et al.. They use CSP as a formulism, and PVS as the theorem prover. The other group, with members of G. Bella and L. Paulson, uses Isabelle/HOL. Both of them did lots of, and similar works.

Steve Schneider et al. introduced the idea of using *rank functions* [102], assigning a rank to each action of principals and intruders, with intention that only actions with strictly positive rank could ever circulate within the system. The ranks that were assigned depend on the protocol itself, the initial knowledge and the capabilities of intruders. They implemented the approach by PVS [52] to verify secrecy and authentication properties [102]. Later, they extended previous model to verify non-repudiation and fairness properties of full ZG protocol based on CSP [103], using a deductive system to describe messages sent by a dishonest principal, and *failures* of a process to define these properties. We borrow the idea of the dishonest principal description from his research. Then they analyzed anonymity property for the dining cryptographers and some variants [104]. Furthermore, they tried to verify the authentication property (a different authentication property from the property in Chapter 6) for the recursive authentication protocol in bounded sessions [34]. For details of this approach, please refer to their book [100].

G. Bella and L. Paulson proposed an inductive verification of security protocols based on trace analysis, and adopted Isabelle/HOL to prove the correctness of various protocols. Firstly, they focused on authentication [95]. Then extended their approach to prove that the full ZG protocol satisfies both non-repudiation and fairness properties [19, 20]. ([20] corrected a bug in [19]). Furthermore, They proved that the RA protocol holds authentication property with bounded number of principals [94]. The property they adopted is also different from our authentication property. For details of this approach, please refer to G. Bella's thesis [18].

The following are other security protocol analysis approaches based on theorem proving.

Kazuhiro Ogata et al. proposed a method for analyzing security protocols on rewriting [90]. A security protocol was modeled as an *observational transition system* (referred to as OTS) [92]. CafeOBJ was adopted to verify the correctness of the protocol. This methodology has later been applied to verify several practical protocols [91, 71].

Yoshinobu Kawabe et al. adopted I/O automata to describe security protocols, and developed a technique to verify anonymity of security protocols. They used a theorem prover, named Larch to implemented their technique and proved the anonymity of an e-voting protocol, which is called FOO protocol [69, 70].

Chapter 11

Conclusions and Perspectives

11.1 Thesis Summaries

This thesis investigated security protocol analysis based on an on-the-fly model checking method. An expressive process calculus based model was introduced to represent behaviors of security protocols. The model is flexible and compositional, deductive systems can be inserted to integrate the model, to represent infinite messages intruders or dishonest principals generate, according to different security assumptions.

The model was abstracted to a finite parametric model with sound and complete representative abilities. In addition, security properties were defined by a set of action terms. It was also proved that the properties defined by these action terms in the original model can be checked in its corresponding parametric model. Thus various security properties, such as secrecy, authentication, non-repudiation, fairness were finitely analyzed in the parametric model.

The model checking method was implemented by Maude. Instead of generating a parametric model previously, each property could be checked at the same time the model being generated. Therefore, it was named an on-the-fly model checking method. With the implemented tool, lots of security protocols were tested, either their attacks were detected, or their security properties were guaranteed.

This method contributes to the research area of security protocols by providing values to the following points.

- Various security properties can be analyzed by this method, driven by different security requirements and different assumptions. The analysis method is sound and complete under the certain assumptions. That is, when flaws are not detected, the analyzed property for a protocol is guaranteed under these assumptions. Among them, the non-repudiation property in bounded session, authentication for recursive protocols are first analyzed by model checking methods.

- Protocol-independent specifications for secrecy and authentication properties are proposed. In this approach, the specifications for secrecy and authentication properties can be generated automatically from a protocol description.

11.2 Future Perspectives and Developments

There may exist three directions for the future researches based on this thesis.

11.2.1 More on the Same Direction

Analyzing Other Properties

Currently, by the method in this thesis, well-known security properties were analyzed. The first future work will adopt the method for other properties. For instance, anonymity [104, 69, 70], and other authentication variations (see Appendix A) can be analyzed within the framework.

Another choice is to analyze existing security properties for other kinds of security protocols, such as, fairness for electronic commerce protocols, or certified e-mail protocols, and so on (see Appendix A). The fairness for these protocols have slight differences from what we have defined for fair non-repudiation protocols. In addition, security properties for security protocols with multiply parties, etc.

Translator

Currently, we only implemented an on-the-fly model checking on the parametric model by Maude. A translator that translates a formal protocol description to a Maude source file is designable. It will also contain the automatic transformation approach to generate a security specification for a given security property.

More Efficient Tool

Current implementation is not efficient enough for analyze a more complex, practical protocol. Refinements should be applied to the current tool. For instance, to implement a more efficient unification algorithm (The analysis relies heavily on unification, while the current algorithm is quite time-consuming), and to refactor current analysis steps.

11.2.2 Affiliating to the Resolution Method

With modern developing technologies, new security protocols have less possibilities to have flaws. Thus comparing to detecting an attack, Proving the correctness of a security protocol shows its essential importance. Model checking method can be regarded as a complement when a protocol fails for being proved.

Recently, the resolution method for security protocol analysis is flourishing [22, 23, 25, 5, 12]. This method is extremely efficient to analyze a protocol (the time it takes is only one percent of time taken by our methods, or even much less), with a cost of possible non-termination (these cases are rare).

Our method can be used as a heuristic guide to extract a counterexample when finding a flaw by resolution. It will provide heuristics for the order of applying horn clauses with which the counterexample is obtained.

11.2.3 Analyzing from a Source Code

Currently, formal security protocol analysis starts from an informal flow by flow descriptions, such as,

$$\begin{aligned} A \longrightarrow B : &\quad \{A, N_A\}_{+K_B} \\ B \longrightarrow A : &\quad \{N_A, N_B\}_{+K_A} \\ A \longrightarrow B : &\quad \{N_B\}_{+K_B} \end{aligned}$$

However, a practical protocol is often given in a document like the *RFC documents*, or a program source code. How to analyze the protocol from these descriptions is a challenge research.

An interesting future direction is to analyze a security protocol from a source code. The first step is probably a way to extract its protocol model from the source code. Then by the similar parametric approach, the security protocol can be analyzed similarly. In the research, our automatic transformation to generate security specifications will show its importance.

Bibliography

[1] *AVISPA*. `http://www.avispa-project.org/`.

[2] *Strand Space*. `http://www.mitre.org/tech/strands/`.

[3] Martín Abadi. Secrecy by Typing in Security Protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[4] Martín Abadi and Bruno Blanchet. Secrecy Types for Asymmetric Communication. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'01)*, volume 2030 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[5] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, 2005.

[6] Martín Abadi and Cédric Fournet. Mobile Values, New Names, and Secure Communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

[7] Martín Abadi and Andrew D. 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. ACM Press, 1997.

[8] Martín Abadi and Andrew D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5:267–303, 1998.

[9] Martín Abadi and Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. Technical Report SRC 125, Digital Systems Research Center, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-125.html, 1989.

[10] Martín Abadi and Mark R. Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the 10th annual ACM symposium on Principles of Distributed Computing (PODC'91)*, pages 201–216. ACM Press, 1991.

[11] Luca Aceto and Matthew B. Hennessy. Termination, Deadlock, and Divergence. *Journal of the ACM*, 39(1):147–187, 1992.

[12] Xavier Allamigeon and Bruno Blanchet. Reconstruction of Attacks against Cryptographic Protocols. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 140–154. IEEE Computer Society Press, 2005.

[13] Roberto M. Amadio and Sanjiva Prasad. The Game of the Name in Cryptographic Tables. In *Proceedings of the 1st International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, volume 1702 of *Lecture Notes in Computer Science*, pages 15–26. Springer-Verlag, 1999.

[14] Jesüs Aranda, Cinzia Di Giusto, Catuscia Palamidessi, and Frank D. Valencia. On Recursion, Replication and Scope Mechanisms in Process Calculi. In *Proceedings of the 6th International Symposium on Formal Methods for Components and Objects (FMCO'07)*. Springer-Verlag.

[15] Alessandro Armando. Deliverable D2.1: The High Level Protocol Specification Language. Technical Report IST-2001-39252, http://www.avispa-project.org/delivs/2.1/d2-1.pdf, 2003.

[16] David A. Basin, Sebastian Mödersheim, and Luca Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-based Analysis of Security Protocols. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 335–344. ACM Press, 2003.

[17] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A Symbolic Model Checker for Security Protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[18] Giampaolo Bella. *Inductive Verification of Cryptographic Protocols*. PhD thesis, University of Cambridge, March 2000.

[19] Giampaolo Bella and Lawrence C. Paulson. Mechanical Proofs about a Non-repudiation Protocol. In *Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'01)*, volume 2152 of *Lecture Notes in Computer Science*, pages 91–104. Springer-Verlag, 2001.

[20] Giampaolo Bella and Lawrence C. Paulson. A Proof of Non-repudiation. In *Proceedings of the 9th International Workshop on Security Protocols (SPW'01)*, volume 2467 of *Lecture Notes in Computer Science*, pages 119–125. Springer-Verlag, 2002.

[21] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka. *Handbook of Process Algebra*. Elsevier, 2001.

[22] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society, 2001.

[23] Bruno Blanchet. From Secrecy to Authenticity in Security Protocols. In *Proceedings of the 9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359. Springer-Verlag, 2002.

[24] Bruno Blanchet, Martín Abadi, and Cedric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 331–340. IEEE Computer Society, 2005.

[25] Bruno Blanchet and Andreas Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. *Theoretical Computer Science*, 333(1-2):67–90, 2005.

[26] Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. Technical Report RR-5727, INRIA, 2006.

[27] Roland Bol and Jan Friso Groote. The Meaning of Negative Premises in Transition System Specifications. *Journal of the ACM*, 43(5):863–914, 1996.

[28] Michele Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *Lecture Notes in Computer Science*, pages 667–681. Springer-Verlag, 2001.

[29] Michele Boreale and Maria Grazia Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Cryptographic Protocols. In *Proceedings of the 17th ACM Symposium on Applied Computing (SAC'02)*, pages 281–285. ACM Press, 2002.

[30] Michele Boreale and Rocco De Nicola. A Symbolic Semantics for the π-calculus. *Information and Computation*, 126(1):34–52, 1996.

[31] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for Cryptographic Processes. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS' 99)*, pages 157–166. IEEE Computer Society, 1999.

[32] Marco Bozzano and Giorgio Delzanno. Automatic Verification of Secrecy Properties for Linear Logic Specifications of Cryptographic Protocols. *Journal of Symbolic Computation*, 38(5):1375–1415, 2004.

[33] Stephen H. Brackin. A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop (CSFW'96)*, pages 62–76. IEEE Computer Society Press, 1996.

[34] Jeremy Bryans and Steve Schneider. CSP, PVS and a Recursive Authentication Protocol. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, 1997.

[35] John A. Bull and David J. Otway. The Authentication Protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, UK, 1997.

[36] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. Technical Report SRC 39, Digital Systems Research Center, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-039.html, 1989.

[37] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[38] John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature: Version 1.0. Technical report, Department of Computer Science, University of York, http://www-users.cs.york.ac.uk/~jac/papers/drareviewps.ps, 1997.

[39] Edmund Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. the MIT Press, 1999.

[40] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincolnand, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *Maude Manual (Version 2.2)*. http://maude.cs.uiuc.edu/maude2-manual/, 2005.

[41] Hubert Comon-Lundh and Véronique Cortier. Security Properties : Two Agents are Sufficient. *Science of Computer Programming*, 50:51–71, 2004.

[42] Luca Compagna. *SAT-based Model Checking of Security Protocols*. PhD thesis, Universitá degli Studi di Genova and the University of Edinburgh, 2005.

[43] Patrick Cousot. Abstract Interpretation. *Symposium on Models of Programming Languages and Computation, ACM Computing Surveys*, 28(2):324–328, 1996.

[44] Patrick Cousot. Abstract Interpretation Based Formal Methods and Future Challenges. In *Informatics - 10 Years Back, 10 Years Ahead*, volume 2000 of *Lecture Notes in Computer Science*, pages 138–156. Springer-Verlag, 2001.

[45] Giorgio Delzanno, Javier Esparza, and Jiří Srba. Monotonic Set-Extended Prefix Rewriting and Verification of Recursive Ping-Pong Protocols. In *Proceedings of the 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 415–429. Springer-Verlag, 2006.

[46] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[47] Danny Dolev, Shimon Even, and Richard M. Karp. On the Security of Ping-Pong Protocols. In *Proceedings of the 2nd Annual International Cryptology Conference (CRYPTO'82)*, volume 1440 of *Lecture Notes in Computer Science*, pages 177–186. Springer-Verlag, 1982.

[48] Danny Dolev and Andrew C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[49] Luca Durante, Riccardo Sisto, and Adriano Valenzano. Automatic Testing Equivalence Verification of Spi Calculus Specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2):222–284, 2003.

[50] Nancy A. Durgin. *Logical Analysis and Complexity of Security Protocols*. PhD thesis, Standford University, 2003.

[51] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[52] Bruno Dutertre and Steve Schneider. Using a PVS Embedding of CSP to Verify Authentication Protocols. In *Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'97)*, volume 1275 of *Lecture Notes in Computer Science*, pages 121–136. Springer-Verlag, 1997.

[53] Clemente Galdi and Raffaella Giordano. Certified E-mail with Temporal Authentication: An Improved Optimistic Protocol. In *Proceedings of the First International Conference on Trust and Privacy in Digital Business (TrustBus'04)*, volume 3184 of *Lecture Notes in Computer Science*, pages 181–190. Springer-Verlag, 2004.

[54] Li Gong, Roger M. Needham, and Raphael Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the 11th IEEE Symposium on Security and Privacy (S&P'90)*, pages 234–248. IEEE Computer Society Press, 1990.

[55] Andrew D. Gordon and Alan Jeffrey. Authenticity by Typing for Security Protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 145–159. IEEE Computer Society Press, 2001.

[56] Andrew D. Gordon and Alan Jeffrey. Types and Effects for Asymmetric Cryptographic Protocols. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 77–91. IEEE Computer Society Press, 2002.

[57] Jan Friso Groote. Transition System Specifications with Negative Premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

[58] Joshua D. Guttman and Fábrega Javier Thayer. Protocol Independence through Disjoint Encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Computer Society, 2000.

[59] Joshua D. Guttman, Fábrega Javier Thayer, Jay A. Carlson, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Trust Management in Strand Spaces: a Rely-guarantee Method. In *Proceedings of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 2004.

[60] James Heather, Gavin Lowe, and Steve Schneider. How to Prevent Type Flaw Attacks on Security Protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 255–268. IEEE Computer Society, 2000.

[61] Charles A. R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice Hall, 1985.

[62] H. Hüttel and J. Srba. Recursion VS. Replication in Simple Cryptographic Protocols. In *Proceedings of the 31st Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'05)*, volume 3381 of *Lecture Notes in Computer Science*, pages 175–184. Springer-Verlag, 2005.

[63] Hans Hüttel and Jiří Srba. Recursive Ping-Pong Protocols. In *Proceedings of the 4th International Workshop on Issues in the Theory of Security (WITS'04)*, pages 129–140, 2004.

[64] ISO/IEC13888-1. Information Technology - Security Techniques - Non-repuduation - Part 1: General. 1997.

[65] ISO/IEC13888-2. Information Technology - Security Techniques - Non-repuduation - Part 2: Mechanisms Using Symmetric Techniques. 1997.

[66] ISO/IEC13888-3. Information Technology - Security Techniques - Non-repuduation - Part 3: Mechanisms Using Asymmetric Techniques. 1997.

[67] Florent Jacquemard, Michaël Rusinowitch, and Laurent Vigneron. Compiling and Verifying Security Protocols. In *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'00)*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160. Springer-Verlag, 2000.

[68] Detlef Käehler, Ralf Küsters, and Tomasz Truderung. Infinite State AMC-Model Checking for Cryptographic Protocols. In *Proceedings of the 21th Annual IEEE Symposium on Logic in Computer Science (LICS'07)*, pages 181–192. IEEE Computer Society Press, 2007.

[69] Yoshinobu Kawabe, Ken Mano, Hideki Sakurada, and Yasuyuki Tsukada. Backward Simulations for Anonymity. In *Proceedings of the 6th International IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'06)*, pages 206–220, 2006.

[70] Yoshinobu Kawabe, Ken Mano, Hideki Sakurada, and Yasuyuki Tsukada. Theorem-proving Anonymity of Infinite-state Systems. *Information Processing Letters*, 101(1):46–51, 2007.

[71] Weiqiang Kong, Kazuhiro Ogata, and Kokichi Futatsugi. Algebraic Approaches to Formal Analysis of the Mondex Electronic Purse System. In *Proceedings of the 6th International Conference on Integrated Formal Methods (IFM'07)*, volume 4591 of *Lecture Notes in Computer Science*, pages 393–412. Springer-Verlag, 2007.

[72] Steve Kremer. *Formal Analysis of Optimistic Fair Exchange Protocols*. PhD thesis, Universite Libre de Bruxelles, 2003.

[73] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An Intensive Survey of Fair Non-repudiation Protocols. *Computer Communications*, 25(17):1606–1621, 2002.

[74] Ralf Küsters and Tomasz Truderung. On the Automatic Analysis of Recursive Security Protocols with XOR. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 646–657. Springer-Verlag, 2007.

[75] Ralf Küsters and Thomas Wilke. Automata-based Analysis of Recursive Cryptographic Protocols. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS'04)*, volume 2996 of *Lecture Notes in Computer Science*, pages 382–393. Springer-Verlag, 2004.

[76] Francesca Levi and Davide Sangiorgi. Mobile Safe Ambients. *ACM Transactions on Programming Languages and Systems*, 25(1), 2003.

[77] Guoqiang Li and Mizuhito Ogawa. On-the-fly Model Checking of Fair Non-repudiation Protocols. In *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *Lecture Notes in Computer Science*, pages 511–522. Springer-Verlag, 2007.

[78] Guoqiang Li and Mizuhito Ogawa. On-the-fly Model Checking of Security Protocols and Its Implementation by Maude. *IPSJ Transactions on Programming*, 48, SIG 10(PRO 33):50–75, 2007.

[79] Gavin Lowe. An Attack on the Needham-Schroeder Public-key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[80] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-key Using FDR. In *Proceedings of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[81] Gavin Lowe. Some New Attacks upon Security Protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop (CSFW'96)*, pages 162–169. IEEE Computer Society Press, 1996.

[82] Gavin Lowe. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.

[83] Gavin Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 18–30. IEEE Computer Society Press, 1997.

[84] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[85] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I/II. *Journal of Information and Computation*, 100:1–77, 1992.

[86] David Monniaux. Abstracting Cryptographic Protocols with Tree Automata. *Science of Computer Programming*, 47(2-3):177–202, 2003.

[87] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[88] Dan M. Nessett. A Critique of the Burrows, Abadi and Needham Logic. *ACM SIGOPS Operating Systems Review*, 24(2):35–38, 1990.

[89] Rocco De Nicola and Matthew B. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.

[90] Kazuhiro Ogata and Kokichi Futatsugi. Rewriting-based Verification of Authentication Protocols. *Electronic Notes in Theoretical Computer Science*, 71:208–222, 2002.

[91] Kazuhiro Ogata and Kokichi Futatsugi. Formal Verification of the Horn-Preneel Micropayment Protocol. In *Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2575 of *Lecture Notes in Computer Science*, pages 238–252. Springer-Verlag, 2003.

[92] Kazuhiro Ogata and Kokichi Futatsugi. Proof Scores in the OTS/CafeOBJ Method. In *Proceedings of the 6th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'03)*, volume 2884 of *Lecture Notes in Computer Science*, pages 170–184. Springer-Verlag, 2003.

[93] Catuscia Palamidessi and Frank D. Valencia. Recursion VS. Replication in Process Calculi: Expressiveness. *Bulletin of the EATCS Column: Concurrency*, 87.

[94] Lawrence C. Paulson. Mechanized Proofs for a Recursive Authentication Protocol. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 84–95. IEEE Computer Society Press, 1997.

[95] Lawrence C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

[96] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[97] Benjamin C. Pierce. *Advanced Topics in Types and Programming Languages*. The MIT Press, 2004.

[98] Peter Ryan and Steve Schneider. An Attack on a Recursive Authentication Protocol. *Information Processing Letters*, 65(1):7–10, 1998.

[99] Peter Ryan and Steve Schneider. Process Algebra and Non-interference. In *Proceedings of the 12th IEEE workshop on Computer Security Foundations (CSFW '99)*, pages 214–227. IEEE Computer Society, 1999.

[100] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley, 2001.

[101] Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2003.

[102] Steve Schneider. Verifying Authentication Protocols with CSP. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 3–17. IEEE Computer Society Press, 1997.

[103] Steve Schneider. Formal Analysis of a Non-repudiation Protocol. In *Proceedings of the 11th Computer Security Foundations Workshop (CSFW'98)*, pages 54–65. IEEE Computer Society Press, 1998.

[104] Steve Schneider and Abraham Sidiropoulos. CSP and Anonymity. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS '96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218. Springer-Verlag, 1996.

[105] Stefan Schwoon. *Model-Checking Pushdown System*. PhD thesis, Technical University of Munich, 2000.

[106] Vitaly Shmatikov and John C. Mitchell. Finite-state Analysis of Two Contract Signing Protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.

[107] Dawn Xiaodong Song. Athena: A New Efficient Automatic Checker for Security Protocol Analysis. In *Proceedings of the 12th Computer Security Foundations Workshop (CSFW'99)*, pages 192–202. IEEE Computer Society, 1999.

[108] William Stallings. *Data & Computer Communications (6th Edition)*. Prentice Hall, 1999.

[109] Paul F. Syverson and Iliano Cervesato. The Logic of Authentication Protocols. In *Proceedings of the Foundations of Security Analysis and Design I (FOSAD'00)*, volume 2171 of *Lecture Notes in Computer Science*, pages 63–136. Springer-Verlag, 2000.

[110] Paul F. Syverson and Paul C. van Oorschot. On Unifying Some Cryptographic Protocol Logics. In *Proceedings of the 15th IEEE Symposium on Security and Privacy (S&P'94)*, pages 14–28. IEEE Computer Society Press, 1994.

[111] Tomasz Truderung. Selecting Theories and Recursive Protocols. In *Proceedings of the 16th International Conference of Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 217–232. Springer-Verlag, 2005.

[112] Christoph Weidenbach. Towards an Automatic Analysis of Security Protocols in First-Order Logic. In *Proceedings of the 16th International Conference on Automated Deduction (CADE'99)*, volume 1632 of *Lecture Notes in Computer Science*, pages 314–328. Springer-Verlag, 1999.

[113] Thomas Y.C. Woo and Simon S. Lam. A Semantic Model for Authentication Protocols. In *Proceedings of the 14th IEEE Symposium on Security and Privacy (S&P'93)*, pages 178–194. IEEE Computer Society Press, 1993.

[114] Thomas Y.C. Woo and Simon S. Lam. A Lesson on Authenticated Protocol Design. *Operating Systems Review*, 28(3):24–37, 1994.

[115] Jianying Zhou. *Non-repudiation in Electronic Commerce*. Computer Security Series. Artech House, 2001.

[116] Jianying Zhou and Dieter Gollmann. A Fair Non-repudiation Protocol. In *Proceedings of the 17th IEEE Symposium on Security and Privacy (S&P'96)*, pages 55–61. IEEE Computer Society Press, 1996.

[117] Jianying Zhou and Dieter Gollmann. Towards Verification of Non-repudiation Protocols. In *Proceedings of 1998 International Refinement Workshop and Formal Methods Pacific*, pages 370–380. Springer-Verlag, 1998.

Publications

[1] Guoqiang Li, Mizuhito Ogawa. On-the-fly Model Checking of Fair Non-repudiation Protocols. In Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07), LNCS 4762, 511-522, 2007

[2] Guoqiang Li, Mizuhito Ogawa. On-the-fly model checking of security protocols and its implementation by Maude. IPSJ Transactions on Programming, Vol.48, No. SIG 10 (PRO 33), 50-75, June, 2007

[3] Min Zhang, Guoqiang Li, Yuxi Fu. Secrecy of Signals by Typing in Signal Transduction. In Proceedings of the 2nd International Conference on Natural Computation (ICNC'06), LNCS 4222, 384-393, 2006

[4] Yonggen Gu, Yuxi Fu, Guoqiang Li. A Simple Process Calculus for the Analysis of Security Protocols. In Proceedings of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05), IEEE Computer Society, 110-114, 2005

[5] Min Zhang, Guoqiang Li, Yuxi Fu, Zhizhou Zhang, Lin He. Typing Aberrance in Signal Transduction. In Proceedings of the 1st International Conference on Natural Computation (ICNC'05), LNCS 3612, 668-677, 2005

Appendix A

A Brief Introduction to Security Protocols

A.1 Security Protocols

Literally, a *security protocol* is a finite sequence of *steps* taken between two or more *protocol roles* using cryptography to establish security properties in a potentially hostile environment. Protocol roles comprise *sender* (also named *originator*), *receiver* (also named *recipient*) and *server* (also named *trusted third party*), and so on. We refer to instances of protocol roles that are participating in an execution as *principals* (also named *agents*). Each principal is instantiated with an identity, for instance, A, B, S. A *session* of a security protocol is one execution of the protocol attended by minimal number of principals. A principal can play multiple roles in different sessions, for example, A may be running the protocols as a sender in one session and a receiver in another session. A *run* is a subsequences of messages sent and received by attended principals.

According to different security aims and goals, security protocols can roughly classify as following two categories, authentication protocols and fair exchange protocols.

A.1.1 Authentication Protocols

Authentication protocols [1] are the most widely used security protocols. An authentication protocol is a sequence of message exchanges between principals, which either distributes secret messages to some of those principals, or allows the use of secret messages to be recognized.

The goals of authentication protocols are to provide various security services across a distributed system. These goals include: establishing session keys between principals, distributing confidential information to expected principals, guaranteeing authentication of principals, ensuring secrecy, integrity, anonymity, and so on. They involve the exchange of information between principals, sometimes requiring the participation of a server. The information they exchange is typically protected by various cryptographic mechanisms, such as symmetric and asymmetric encryptions, one-way hash functions and so on. In some cases further devices like timestamps are also used.

A considerable number of authentication protocols have been specified and implemented. However, many protocols have been shown to be flawed a long time after they

[1] For intensive surveys and discussions of authentication protocols, referred to [36, 38].

were published. A well-known example is the Needham-Schroeder authentication protocol. The Needham-Schroeder symmetric key protocol (referred to as the NSSK protocol) was published in 1978 [87] and became the basis for many similar protocols in later years. The authors suggested an alternative protocol based on public key cryptography, which is named the Needham-Schroeder public key protocol (referred to as the NSPK protocol). In 1981, Denning and Sacco demonstrated that the Needham-Schroeder symmetric key Protocol was flawed and proposed a refined protocol [46]. In 1994, Martín Abadi demonstrated that the protocol that Denning and Sacco refined was also flawed [9]. Furthermore, In 1995, Lowe found an attack on the NSPK protocol (17 years after its publication) [79, 80].

Other authentication protocols also have flaws, which are not easily to be detected. Here are some: the Woo-Lam protocol, the Yahalom protocol, the Otway-Rees protocol, etc.. They will be analyzed in this thesis.

A.1.2 Fair Exchange Protocols

A fair exchange protocol [2] deals with the exchange of two items of a given value between two principals (A and B, for example). The main aim is to realize this exchange without the risk of each principals being disadvantaged, e.g., A does obtain B's item, but B does not obtain A's item.

There are three general constructions for fair exchange protocols, based on the degree of the involvement of TTP. The first class, which chronologically precedes the other classes, are fair exchange protocols with no TTPs. These are based on gradual release of information and require exchanging many messages to "approximate" fair exchange (fair exchange protocols with no TTPs are theoretically impossible). Moreover, they often assume that the principals have equal computational powers. Protocols of the second class need the TTP's intervention in each exchange. A drawback of these protocols is that the TTP easily becomes a communication bottleneck or a single target of attacks. The third class of fair exchange protocols, known as *optimistic fair exchange protocols* [72], requires the TTP's intervention only if a failure (accidentally or maliciously) occurs. Therefore, honest parties that are willing to exchange their items can do so without involving any TTP.

Fair exchange protocols classifies a vast class of protocols, with slightly different aims of *fairness*.

Electronic Commerce Protocols

Electronic commerce protocols (referred to as EC protocols) is one of the most natural applications of fair exchange protocols. An EC protocol is combined with some of subprotocols, like an electronic payment protocol, an electronic management protocol, and so on. Due to extremely importance of EC protocols, They almost require all security goals, such as authentication, secrecy, non-repudiation, fairness, integrity, anonymity, and so on.

[2] For intensive surveys and historical overviews of fair exchange protocols, referred to [72, 73].

Non-repudiation Protocols

Fair non-repudiation protocols [64, 65, 66, 115] intend reliable exchange of messages in the situation that each principal can be dishonest, who tries to take advantages from other principals by aborting the communication or sending fake messages. The main goal of a non-repudiation protocol is to produce the evidences for the sender and the receiver respectively. An evidence is used digital signatures technology, so that a principal cannot deny sending the message with its signatures. Non-repudiation protocols should guarantee that it is impossible to reuse the same evidence for different messages or principals.

In this thesis, both non-repudiation and fairness properties of fair non-repudiation protocols are analyzed by our methodology.

Certified E-mail Protocols

Certified E-mail protocols (referred to as CEM protocols) are to provide such services: A wants to send an E-mail to B. It wishes to receive a receipt when B receives (and is able to read) the email. According to different security goals, sometimes B is willing to send back a receipt to A only if he guarantee the E-mail comes from A (which is similar to non-repudiation protocols); sometimes B must successfully send his receipt without getting any A's information (Thus the protocols should satisfy authentication and reputation properties).

Digital Contract Signing Protocols

Digital contract signing protocols are ones of the simplest forms of fair exchange protocols. In a digital contract signing protocol, A and B want to exchange their corresponding signatures on a given contract text, which is known by both two principals, which makes the protocol design easier.

A.2 Security Properties

A.2.1 Secrecy

Secrecy, or confidentiality, is a basic goal of security protocols. Intuitively, the secrecy property means that an intruder is not able to derive a confidential datum that the principals communicate. This thesis takes this definition. Furthermore, there still exists a strict interpretation of secrecy property. That is, an intruder should not be able to perform any traffic analysis without knowing the exact confidential datum. This kind of secrecy is also named *non-interference*, which is required that a high-level user's activity should not result in any observable effect on low-level users or outside observers of the system [3, 99].

A.2.2 Authentication

The *authentication* property is another important security property that is usually studied in security protocol analysis. Intuitively, authentication means that it can be sure that a message that purports to be from a certain principal was indeed originated

by that principal. Gavin Lowe had further discussed the definition of authentication [82], which has been hierarchized as follows:

Aliveness A protocol guarantees A *aliveness* of B if, whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol.

Weak agreement A protocol guarantees A *weak agreement* with B if, whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A.

Non-injective agreement A protocol guarantees A *non-injective agreement* with B on a set of data ds if, whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A, and two principals agreed on the data values corresponding to all the variables in ds.

Agreement A protocol guarantees A *agreement* with B on a set of data ds if, whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A, and two principals agreed on the data values corresponding to all the variables in ds, and each such run of A corresponds to a unique run of B.

Almost all papers [80, 13, 28, 17, 78] took non-injective agreement explained above as authentication properties. This thesis also follows this explanation. And by our action terms, other kinds of authentication can also be defined and analyzed.

A.2.3 Non-repudiation

For most security goals, we assume that legitimate principals are honest, i.e., behave according to the protocol rules. The aim of *non-repudiation*, is to protect one principal against possible cheating by other principals. Thus, each principal can be dishonest, trying to take advantages from other principals by aborting the communication or sending fake messages. Under this assumption, non-repudiation is when a sender sends some message to a receiver, neither the sender nor the receiver can deny this after participating in this communication. Usually, it concerns the following two properties [115, 73]:

- *Non-repudiation of origin (NRO)* is intended to protect against the sender's false denial of having sent the messages.
- *Non-repudiation of receipt (NRR)* is intended to protect against the receiver's false denial of having received the message.

This thesis takes the above two definitions as non-repudiation properties. Furthermore, when we consider indirect communication model, in which a delivery agent is involved to transfer a message from two principals, non-repudiation should further concern the following two properties [115].

- *Non-repudiation of submission (NRS)* is intended to provide the evidence that the sender submitted the message for delivery.
- *Non-repudiation of delivery (NRD)* is intended to provide that the message has been delivered to the receiver.

A.2.4 Fairness

Similar to non-repudiation, fairness should also be defined under the assumption that each principal can be dishonest. Fairness means no principals can obtain an important item from other principal while the other principal cannot do so. The important item varies from different protocols. For example, in a non-repudiation protocol, the item means the evidence of the counterpart principal; in a certified E-mail Protocols, items means the Email and reader's receipt, respectively, etc. For detailed summarizations and comparisons of fairness definitions, referred to [72].

A.3 Vulnerabilities and Attacks

To illustrate kinds of attacks for security protocols shows us strategies that intruders might employ. Note that our method does not depend on knowing these strategies.

The attacks we introduce blow are of course not exhaustive. They will serve to illustrate the various styles of attack. There are still other styles of attacks, such as cryptanalytic, monitoring timing, or fluctuations in power consumption. These attacks are out the scope of our analysis.

A useful categorization of attacks is in terms of *passive attacks* and *active attacks* [108].

A.3.1 Passive Attacks

Passive attacks are in the nature of eavesdropping on transmissions of networks. These attacks often violate secrecy of a security protocol, since they can get the confidential information from a public network. Basically, there are two kinds of passive attacks.

Releasing Messages

This kind of attacks is easily understood. Intruders may obtain messages from networks, then analyze them based on their knowledge and computation ability to get confidential information. It obviously violates the secrecy property. So we should prevent intruders from learning the information from transmissions.

Traffic Analysis

Traffic Analysis (also named as *oracle attack*) is more subtle. Suppose we encrypt each message with a "perfect" encryption system, so that intruders could not extract the information from these messages, even if they captured the messages. However, intruders might still be able to observe the pattern of these messages. They could determine the location and identity of communicating hosts, and observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of communications. One possible attack of traffic analysis is to violate the non-interference property [3, 99].

A.3.2 Active Attacks

Active attacks involve some modifications of the data stream, or the creation of a false stream. It has many categories. We only introduce some of them in the following

subsections. Note that an attack to a security protocol may be composed of several kinds of attacks introduced below. For instance, almost all attacks for protocols in multiple sessions are composed of man-in-middle attacks and replay attacks.

Man-in-middle Attack

Man-in-middle attacks involve an intruder imposing himself in the communications between two principals in various ways. For instance, he may be able to masquerade as principal to the other principal. The most well-known man-in-middle attack may be the attack for the NSPK protocol (see Subsection 5.5.2). An intruder communicates with A, and at the same time masquerades A to communicate with B.

Man-in-middle attacks are difficult to detect, since usual models only model that each honest principal communicates with each other. Our method is efficient, since by binders we parameterize principals one may communicate. Thus we can detect the attack for the NSPK protocol in one session, while other method detected it in two sessions. In their methodologies, a principal will explicitly communicate with an intruder in one session [17, 28].

Replay Attack

In a *replay attack*, intruders monitor a run of the protocol, and at some later communications they replays one or more of the messages. If a security protocol does not have any mechanism (like nonces, timestamps, for instance) to distinguish between two separate sessions, or it cannot detect the staleness of a message, it is quite possible to fool honest principals into rerunning parts of the protocol. An example of replay attacks is the attack for the simplified ZG protocol (see Subsection 7.4). An intruder uses the message sent by A in the first flow to cheat the server in the third flow. In comparison, in the full ZG protocol, a set of nonces is adopted, and thus replay attacks are avoided.

Reflection Attack

Reflection attacks are to send messages that a principal generated back to itself. Sometimes this can fool the sender into revealing the correct response to its messages. The attacks usually happen in the symmetry of the situation. The general attack outline is as follows:

- A intruder initiates a protocol to a server.
- The server attempts to authenticate the intruder by sending it a message.
- The intruder opens another connection to the target, and sends the server this message as its own.
- The server responds to the message.
- The intruder sends that response back to the server on the original connection. If the protocol is not carefully designed, the target will accept that response as valid, thereby leaving the intruder with an authenticated communication.

Denial of Service Attack

A *denial of service attack* (DoS attack) is an attempt to make a computer resource unavailable to its intended users. Intruders of DoS attacks typically, but not exclusively, target sites or services hosted on high-profile web servers.

One common method of attack involves saturating the target machine with external communications requests, so that it cannot respond to legitimate traffic, or responds so slowly as to be rendered effectively unavailable. In general terms, DoS attacks are implemented by,

- forcing the targeted computer(s) to reset, or consume its resources such that it can no longer provide its intended service.
- obstructing the communication media between the intended users and the target machine so that they can no longer communicate adequately.

Appendix B

Semantics of Process Calculi

This chapter will briefly introduce several different semantics of process calculi. These semantics are adopted by various process calculi. The purpose of this chapter is to refer to these semantics under a unified formulism, in which these terminologies are used in this thesis, one can have an intuitive understanding of them. Firstly, CCS will be chosen as the first calculus, introducing transitional labeled transitional semantics, trace semantics, failure semantics and testing semantics. Then π-calculus will be adopted as a value-passing calculus to illustrate early semantics, late semantics and symbolic semantics.

B.1 CCS

The CCS introduced in this section comes originally from [84]. We shall assume an infinite set $\mathcal{A}$ of *names*, and use $a, b, c, \ldots$ to range over A. We denote by $\overline{\mathcal{A}}$ the set of co-name. $\overline{a}, \overline{b}, \overline{c}, \ldots$ will range over $\overline{\mathcal{A}}$. Then we let $\mathcal{L}$, the set of *labels*, satisfies $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$, and define set of actions, $Act = \mathcal{L} \cup \{\tau\}$. We let $\alpha, \beta, \ldots$ range over Act.

Further, we introduce a set $\mathcal{K}$ of *agent constants*. Let $A, B, \ldots$ range over $\mathcal{K}$. Then the set of agent is denoted by $\mathcal{P}$, and we shall let $P, Q, \ldots$ range over agents. Then $\mathcal{P}$ can be inductively defined as follows:

$$P, Q, R ::= \mathbf{0} \mid \alpha.P \mid P + Q \mid P|Q \mid P\backslash L \mid P[f] \mid A$$

A constant is an agent whose meaning is given by a unique defining equation. We assume for every constant A, there is a defining equation of the form,

$$A \triangleq P$$

B.1.1 Transitional Semantics

For the given set of agents, we will use the general notion of a *labeled transition system*

$$\langle S, T, \{\xrightarrow{t} : t \in T\}\rangle$$

which consists of a set S of *states*, a set T of *transition labels* and a transition relation $\xrightarrow{t} \subseteq S \times S$ for each $t \in T$.

The complete set of transitions rules is given in Figure B.1; the name Act, Sum, Com, Res, RelCon indicate that the rules are associated with prefix, summation, composition, restriction, relabeling and with constant, respectively.

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \textsc{Act} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ \textsc{Suml} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}\ \textsc{Sumr}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{Coml} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \textsc{Comr}$$

$$\frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textsc{Com}\tau$$

$$\frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L}(\alpha, \overline{\alpha} \notin L)\ \textsc{Res} \qquad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}\ \textsc{Rel}$$

$$\frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}(A \triangleq P)\ \textsc{Con}$$

Figure B.1: Labeled Transition System of CCS

B.1.2 Strong and Weak Bisimulations

This subsection introduces a common algebraic theory of process calculi, *bisimulation*. Two fundamental definitions of bisimulations for CCS are given, and these lead to notions of equalities over the calculus, called *strong/observation equivalence* or *strong/weak bisimilarity*, respectively. They are important semantical equivalences over CCS processes. The notion of bisimulation appears in many places in mathematical logic and computer science under different names (Note that the following definitions mainly come from [84]).

Definition B.1 (Strong bisimulation)**.** A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a strong bisimulation if $(P, Q) \in \mathcal{S}$ implies, for all $\alpha \in Act$,

- Whenever $P \xrightarrow{\alpha} P'$, then for some Q', $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$;
- Whenever $Q \xrightarrow{\alpha} Q'$, then for some P', $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$;.

Definition B.2 (Strong equivalence)**.** P and Q are strongly equivalent or strongly bisimilar, written $P \sim Q$, if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. That is,

$$\sim = \bigcup\{\mathcal{S} \mid \mathcal{S} \text{ is a strong bisimulation}\}$$

The following bisimulation is relaxed to allow some, but not all, of the internal behaviors of a system to be ignored, which leads to a notion of observation equivalence. Before defining the bisimulation, several preliminary definitions are needed.

Definition B.3 (Preliminary definitions)**.** If $t = \alpha_1 \ldots \alpha_n \in Act^*$, then

- $\hat{t} \in \mathcal{L}^*$ is the sequence gained by removing all occurrences of τ from t.
- We write $P \xrightarrow{t} P'$, if $P \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P'$. We also write $P \xrightarrow{t}$ to mean that $P \xrightarrow{t} P'$ for some P'.

- $P \stackrel{t}{\Longrightarrow} P'$, if $P \stackrel{\tau}{\longrightarrow}^* \stackrel{\alpha_1}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* \ldots \stackrel{\tau}{\longrightarrow}^* \stackrel{\alpha_n}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* P'$. We also write $P \stackrel{t}{\Longrightarrow}$, if $P \stackrel{t}{\Longrightarrow} P'$, for some P'

Definition B.4 (Weak bisimulation)**.** A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a (weak) bisimulation if $(P, Q) \in \mathcal{S}$ implies, for all $\alpha \in Act$,

- Whenever $P \stackrel{\alpha}{\longrightarrow} P'$, then for some Q', $Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{S}$;
- Whenever $Q \stackrel{\alpha}{\longrightarrow} Q'$, then for some P', $P \stackrel{\hat{\alpha}}{\Longrightarrow} P'$ and $(P', Q') \in \mathcal{S}$.

Definition B.5 (Observation equivalence)**.** P and Q are observation-equivalent or weakly bisimilar, written $P \approx Q$, if $(P, Q) \in \mathcal{S}$ for some weak bisimulation $\mathcal{S}$. That is,

$$\approx = \bigcup \{\mathcal{S} \mid \mathcal{S} \text{ is a weak bisimulation}\}$$

B.1.3 Trace Semantics and Equivalence

Besides strong and weak bisimilarity, some variations of equivalences are also proposed, abstracting internal actions in different ways, in which trace equivalence is perhaps the simplest and most straightforward.

Trace semantics explicitly records actions performed by a process with respect to some transition by a *trace*. The configuration of a trace semantics is usually denoted by a pair $\langle s, P \rangle$, in which s is a trace, and P is a process. Note that according to Definition B.3, $\langle s, P \rangle$ with respect to $\longrightarrow$ is represented by $P \stackrel{t}{\longrightarrow}$. Thus we have trace equivalence.

Definition B.6 (Trace equivalence)**.** P and Q are trace equivalent, written $P \approx_t Q$, if for all $s \in \mathcal{L}^*$, $P \stackrel{s}{\Longrightarrow}$ if and only if $Q \stackrel{s}{\Longrightarrow}$.

B.1.4 Failure Semantics and Equivalence

Failure equivalence appears between trace equivalence and weak bisimilarity. The following definition of *failure* comes from Professor Yuxi Fu's lecture, who pointed out that the same definition in [84] was flawed.

Definition B.7 (Failure)**.** A failure is a pair (s, L), where $s \in \mathcal{L}^*$ is a trace and $L \subseteq \mathcal{L}$ is a set of labels. The failure (s, L) is said to belong to a process P if there exists P' such that

- $P \stackrel{s}{\Longrightarrow} P'$
- For all $l \in L$, $P' \stackrel{l}{\not\Longrightarrow}$

Definition B.8 (Failure Equivalence)**.** Two processes P and Q are failures-equivalent, written $P \approx_f Q$, if they possess exactly the same failures.

B.1.5 Testing Semantics and Equivalence

Testing semantics is another semantic theory for processes. In the theory, the behavior of programs or processes can be investigated by a series of *tests*. For sequential programs, a test is considered as a pair, consisting of a predicate on the input domain and a predicate on the output domain.

We redefine the following concepts in [89] with modern notations in the unified formulism we used above.

General Definitions

Let's have the following predefinitions:

- A set of *states*, $\mathcal{S}_{\mathcal{T}}$, and let s_t range over states;
- A *computation* is any non-empty sequence of states. Let $\mathcal{C}_{\mathcal{T}}$ denote the set of computations, ranged over by c_t.
- Let $\mathcal{O}_{\mathcal{T}}$, $\mathcal{P}_{\mathcal{T}}$ (ranged over by o_t, p_t) be sets of *observers* and *processes*.
- An observer o_t performing tests on processes p_t is denoted by a non-empty set of computations, $\mathcal{C}_{\mathcal{T}}(o_t, p_t)$. If $c_t \in \mathcal{C}_{\mathcal{T}}(o_t, p_t)$, we say that the result of o_t testing p_t may be the computation c_t.
- A set $\mathcal{S}_{suc} \subseteq \mathcal{S}_{\mathcal{T}}$ is a set of *success*. A computation is *successful* if it contains a successful state. Otherwise, it is called *unsuccessful*.
- For each state, there exists a unary post-fixed predicate, $\uparrow$.

Definition B.9 (Divergence)**.** Divergence is a unary post-fixed predicate on computations, written $\Uparrow$. $c_t \Uparrow$ if

- c is unsuccessful.
- c contains a state s, such that $s \uparrow$, and is not preceded by a successful state.

Definition B.10 (Result)**.** For every $o_t \in \mathcal{O}_{\mathcal{T}}$, $p_t \in \mathcal{P}_{\mathcal{T}}$, let the result, $\mathcal{R}_{\mathcal{T}}(o_t, p_t) \subseteq \{\top, \bot\}$ be defined by,

- $\top \in \mathcal{R}_{\mathcal{T}}(o_t, p_t)$, if $\exists c_t \in \mathcal{C}_{\mathcal{T}}(o_t, p_t)$ such that c_t is successful.
- $\bot \in \mathcal{R}_{\mathcal{T}}(o_t, p_t)$, if $\exists c_t \in \mathcal{C}_{\mathcal{T}}(o_t, p_t)$ such that $c_t \Uparrow$.

The set $\{\top, \bot\}$ can be viewed as a two point lattice (see (a), Figure B.2). By the theory of power domains, three different subsets are constructed to give three different orderings on result sets, given in Figure B.2, (b), (c), (d), which are denoted by $\sqsubseteq_1$, $\sqsubseteq_2$, $\sqsubseteq_3$, respectively.

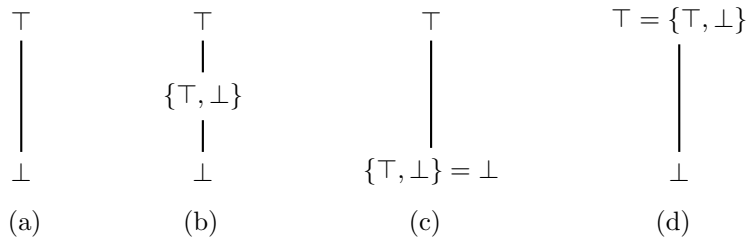


Figure B.2: Lattice on Result Sets

Definition B.11 (Testing Equivalences)**.** For given sets of observers and processes, $\mathcal{O}_\mathcal{T}$ and $\mathcal{P}_\mathcal{T}$, respectively, let $\sqsubseteq_i^{\mathcal{O}_\mathcal{T}}$, $i = 1, 2, 3$ be defined by,

$$p_t \sqsubseteq_i^{\mathcal{O}_\mathcal{T}} q_t \text{ if } \forall o_t \in \mathcal{O}_\mathcal{T}, \mathcal{R}_\mathcal{T}(o_t, p_t) \sqsubseteq_i \mathcal{R}_\mathcal{T}(o_t, q_t)$$

Testing equivalences over a set of observers $\mathcal{O}_\mathcal{T}$, denoted by $\simeq_i^{\mathcal{O}_\mathcal{T}}$, are relations such that $\simeq_i^{\mathcal{O}_\mathcal{T}} = \sqsubseteq_i^{\mathcal{O}_\mathcal{T}} \cap \sqsupseteq_i^{\mathcal{O}_\mathcal{T}}$.

If $\top \in \mathcal{R}_\mathcal{T}(o_t, p_t)$, we say that p_t *may satisfy* o_t. If $\top = \mathcal{R}_\mathcal{T}(o_t, p_t)$, we say that p_t *must satisfy* o_t.

Testing Semantics for CCS

The following shows that how to view CCS as a particular example of the general setting of testing semantics. Firstly, let's define a set of predicate on closed processes of CCS.

Definition B.12 (Predicate on CCS)**.** Let $\downarrow$ be the least predicate on closed processes which satisfies:

- $\mathbf{0} \downarrow$, $\alpha.P \downarrow$.
- $P \downarrow$, $Q \downarrow$ imply $(P + Q) \downarrow$, $(P \mid Q) \downarrow$, $(P \backslash L) \downarrow$, and $(P[f]) \downarrow$.
- $A \downarrow$ if $A \triangleq P$ and $P \downarrow$.

Let $P \uparrow$ if not $P \downarrow$. For instance, $A \uparrow$ in $A \triangleq c.\mathbf{0} + A$. Intuitively, $P \uparrow$ means that there is an unguarded occurrence of a constant.

The set of $\mathcal{P}_\mathcal{T}$ contains the closed processes in CCS, while the set of $\mathcal{O}_\mathcal{T}$ contains all closed processes in CCS, in which set of actions is $Act^* = Act \cup \{\omega\}$. ω is a distinguished action symbol, not in Act. It is used to report success.

Moreover,

- $\mathcal{S}_\mathcal{T} = \mathcal{O}_\mathcal{T}$,
- $\mathcal{S}_{suc} = \{P \mid \exists P'.P \xrightarrow{\omega} P'\}$
- A computation is any sequence of terms $\{P_n \mid n \geq 0\}$ such that, if P_n is final element in the sequence then $P_n \overset{\tau}{\not\longrightarrow}$, otherwise $P_n \xrightarrow{\tau} P_{n+1}$.
- $\mathcal{C}_\mathcal{T}(o_t, p_t)$ are the set of computations whose initial element is the process $(o_t \mid p_t)$.

B.2 π-calculus

π-calculus is used to illustrate value-passing process calculi, by which three traditional semantics are introduced. The syntax of π-calculus adopted here comes originally from [101]. Note that the following definitions are sightly different from those defined in [85].

Definition B.13 (Prefixes)**.** Let $\mathcal{N}$ be a countable set of names, and let x, y range over it. The capabilities for actions of the calculus are expressed via prefixes, of which there are four kinds:

$$\pi ::= \overline{x}y \mid x(z) \mid \tau \mid [x = y]\pi$$

Definition B.14 (Processes)**.** The processes of the π-calculus are given by

$$\begin{aligned} P ::= &\quad M \mid P|P \mid (\nu z)P \mid !P \\ M ::= &\quad \mathbf{0} \mid \pi.P \mid M + M \end{aligned}$$

B.2.1 Early Semantics

The content in this subsection comes originally from [101]. The actions for the early labeled transition system are given by

$$\alpha ::= \overline{x}y \mid xy \mid \overline{x}(z) \mid \tau$$

Table B.1 shows terminology and notation for actions in early transition system: the *kind* of the action, its *subject* (channel), its *object* (sending name), its set of *free names*, its set of *bound names*, its set of *names*, and the effect of applying a substitution to it.

α	kind	$\mathtt{subj}(\alpha)$	$\mathtt{obj}(\alpha)$	$\mathtt{fn}(\alpha)$	$\mathtt{bn}(\alpha)$	$\alpha\sigma$
$\overline{x}y$	free output	x	y	$\{x, y\}$	$\emptyset$	$\overline{x\sigma}y\sigma$
xy	input	x	y	$\{x, y\}$	$\emptyset$	$x\sigma y\sigma$
$\overline{x}(y)$	output	x	y	$\{x\}$	$\{y\}$	$\overline{x\sigma}(y)$
τ	internal	_	_	$\emptyset$	$\emptyset$	τ

Table B.1: Terminology and Notation for Actions in the Early Transition System

The early transition relations are defined in the Figure B.3.

Definition B.15 (Early bisimilarity)**.** Early bisimilarity is the largest symmetric relation, $\sim_e$, such that whenever $P \sim_e Q$,

- $P \xrightarrow{x(z)} P'$ implies, for every y there is Q' such that $Q \xrightarrow{x(z)} Q'$ and $P' \sim_e Q'$.
- if α is not an input action then $P \xrightarrow{\alpha} P'$ implies $Q \xrightarrow{\alpha} Q'$ and $P' \sim_e Q'$.

B.2.2 Late Semantics

The content in this subsection also comes from [101]. The actions for the late labeled transition system are given by

$$\alpha ::= \overline{x}y \mid x(y) \mid \overline{x}(z) \mid \tau$$

Table B.2 shows terminology and notation for actions in early transition system.

The main difference between the early and the late transition system is the input action rules, INP and L-INP. In the late relations, the *free input actions* xy are replaced by *bound input actions* $x(z)$.

The late transition relations are defined in the Figure B.4.

On the late semantics, we can also define the early bisimilarity. There is a slight difference from the early bisimilarity defined on the early semantics. Besides that, we can also define a late bisimilarity.

$$\frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}\ \text{OUT} \qquad \frac{}{x(z).P \xrightarrow{xy} P\{y/z\}}\ \text{INP} \qquad \frac{}{\tau.P \xrightarrow{\tau} P}\ \text{TAU}$$

$$\frac{\pi.P \xrightarrow{\alpha} P'}{[x=x]\pi.P \xrightarrow{\alpha} P'}\ \text{MAT}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset\ \text{PAR-L}$$

$$\frac{P \xrightarrow{\alpha} P'}{Q \mid P \xrightarrow{\alpha} Q \mid P'}\ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset\ \text{PAR-R}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P' + Q}\ \text{SUM-L} \qquad \frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} Q + P'}\ \text{SUM-R}$$

$$\frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{COMM-L} \qquad \frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\overline{x}y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{COMM-R}$$

$$\frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}\ z \notin \mathtt{fn}(Q)\ \text{CLOSE-L}$$

$$\frac{P \xrightarrow{xz} P' \quad Q \xrightarrow{\overline{x}(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}\ z \notin \mathtt{fn}(P)\ \text{CLOSE-R}$$

$$\frac{P \xrightarrow{\alpha} P'}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}\ z \notin \mathtt{n}(\alpha)\ \text{RES} \qquad \frac{P \xrightarrow{\overline{x}z} P'}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}\ z \neq x\ \text{OPEN}$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\tau} P' \mid !P}\ \text{REP-ACT} \qquad \frac{P \xrightarrow{\overline{x}y} P' \quad P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}\ \text{REP-COMM}$$

$$\frac{P \xrightarrow{\overline{x}(z)} P' \qquad P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} ((\nu z)(P' \mid P'')) \mid !P}\ z \notin \mathtt{fn}(P)\ \text{REP-CLOSE}$$

Figure B.3: The Early Transition Rules

α	kind	$\mathtt{subj}(\alpha)$	$\mathtt{obj}(\alpha)$	$\mathtt{fn}(\alpha)$	$\mathtt{bn}(\alpha)$	$\alpha\sigma$
$\overline{x}y$	free output	x	y	$\{x, y\}$	$\emptyset$	$\overline{x\sigma}y\sigma$
$x(y)$	bound input	x	y	$\{x\}$	$\{y\}$	$x\sigma(y)$
$\overline{x}(y)$	output	x	y	$\{x\}$	$\{y\}$	$\overline{x\sigma}(y)$
τ	internal	_	_	$\emptyset$	$\emptyset$	τ

Table B.2: Terminology and Notation for Actions in the Late Transition System

Definition B.16 (Early bisimilarity on the late semantics)**.** Early bisimilarity on the late semantics is the largest symmetric relation, $\sim_e^l$, such that whenever $P \sim_e^l Q$,

- $P \xrightarrow{x(z)} P'$ implies, for every y there is Q' such that $Q \xrightarrow{x(z)} Q'$ and $P'\{y/z\} \sim_e^l$

$$\frac{}{\overline{x}y.P \overset{\overline{x}y}{\longmapsto} P}\ \text{L-OUT} \qquad \frac{}{x(z).P \overset{x(z)}{\longmapsto} P}\ \text{L-INP} \qquad \frac{}{\tau.P \overset{\tau}{\longmapsto} P}\ \text{L-TAU}$$

$$\frac{\pi.P \overset{\alpha}{\longmapsto} P'}{[x=x]\pi.P \overset{\alpha}{\longmapsto} P'}\ \text{L-MAT}$$

$$\frac{P \overset{\alpha}{\longmapsto} P'}{P \mid Q \overset{\alpha}{\longmapsto} P' \mid Q}\ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset\ \text{L-PAR-L}$$

$$\frac{P \overset{\alpha}{\longmapsto} P'}{Q \mid P \overset{\alpha}{\longmapsto} Q \mid P'}\ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset\ \text{L-PAR-R}$$

$$\frac{P \overset{\alpha}{\longmapsto} P'}{P + Q \overset{\alpha}{\longmapsto} P' + Q}\ \text{L-SUM-L} \qquad \frac{P \overset{\alpha}{\longmapsto} P'}{Q + P \overset{\alpha}{\longmapsto} Q + P'}\ \text{L-SUM-R}$$

$$\frac{P \overset{\overline{x}y}{\longmapsto} P' \qquad Q \overset{x(z)}{\longmapsto} Q'}{P \mid Q \overset{\tau}{\longmapsto} P' \mid Q'\{y/z\}}\ \text{L-COMM-L} \qquad \frac{P \overset{x(z)}{\longmapsto} P' \qquad Q \overset{\overline{x}y}{\longmapsto} Q'}{P \mid Q \overset{\tau}{\longmapsto} P'\{y/z\} \mid Q'}\ \text{L-COMM-R}$$

$$\frac{P \overset{\overline{x}(z)}{\longmapsto} P' \qquad Q \overset{x(z)}{\longmapsto} Q'}{P \mid Q \overset{\tau}{\longmapsto} (\nu z)(P' \mid Q')}\ \text{L-CLOSE-L}$$

$$\frac{P \overset{x(z)}{\longmapsto} P' \qquad Q \overset{\overline{x}(z)}{\longmapsto} Q'}{P \mid Q \overset{\tau}{\longmapsto} (\nu z)(P' \mid Q')}\ \text{L-CLOSE-R}$$

$$\frac{P \overset{\alpha}{\longmapsto} P'}{(\nu z)P \overset{\alpha}{\longmapsto} (\nu z)P'}\ z \notin \mathtt{n}(\alpha)\ \text{L-RES} \qquad \frac{P \overset{\overline{x}z}{\longmapsto} P'}{(\nu z)P \overset{\overline{x}(z)}{\longmapsto} P'}\ z \neq x\ \text{L-OPEN}$$

$$\frac{P \overset{\alpha}{\longmapsto} P'}{!P \overset{\alpha}{\longmapsto} P' \mid !P}\ \text{L-REP-ACT} \qquad \frac{P \overset{\overline{x}y}{\longmapsto} P' \qquad P \overset{x(z)}{\longmapsto} P''}{!P \overset{\tau}{\longmapsto} (P' \mid P''\{y/z\}) \mid !P}\ \text{L-REP-COMM}$$

$$\frac{P \overset{\overline{x}(z)}{\longmapsto} P' \qquad P \overset{x(z)}{\longmapsto} P''}{!P \overset{\tau}{\longmapsto} ((\nu z)(P' \mid P'')) \mid !P}\ \text{L-REP-CLOSE}$$

Figure B.4: The Late Transition Rules

$Q'\{y/z\}$.

- if α is not an input action then $P \overset{\alpha}{\longmapsto} P'$ implies $Q \overset{\alpha}{\longmapsto} Q'$ and $P' \sim_e^l Q'$.

Definition B.17 (Late bisimilarity)**.** Late bisimilarity is the largest symmetric relation, $\sim_l$, such that whenever $P \sim_l Q$,

- $P \overset{x(z)}{\longmapsto} P'$ implies there is Q' such that $Q \overset{x(z)}{\longmapsto} Q'$ and $P'\{y/z\} \sim_l Q'\{y/z\}$ for every y.
- if α is not an input action then $P \overset{\alpha}{\longmapsto} P'$ implies $Q \overset{\alpha}{\longmapsto} Q'$ and $P' \sim_l Q'$.

B.2.3 Symbolic Semantics

The content in this subsection comes originally from [30], with several modifications.

Definition B.18 (Boolean formulae)**.** Let $\mathcal{N}$ be the set of names, and let x, y range over it. Let ϕ range over the language of boolean formulae, $\mathcal{BF}$.

$$\phi ::= true \mid [x = y] \mid \neg\phi \mid \phi \wedge \phi$$

The symbolic transition relations are defined in the Figure B.5, in which the function $\mathcal{R}_z : \mathcal{BF} \to \mathcal{BF}$:

$$\begin{array}{rcll} \mathcal{R}_z(true) & = & true & \\ \mathcal{R}_z([w_1 = w_2]) & = & [w_1 = w_2] & z \notin \{w_1, w_2\} \\ \mathcal{R}_z([z = z]) & = & true & \\ \mathcal{R}_z([z = w]) & = & false & z \neq w \\ \mathcal{R}_z(\neg\phi) & = & \neg\mathcal{R}_z(\phi) & \\ \mathcal{R}_z(\phi \wedge \psi) & = & \mathcal{R}_z(\phi) \wedge \mathcal{R}_z(\psi) & \end{array}$$

$$\frac{}{\overline{x}y.P \xrightarrow{true,\overline{x}y} P}\ \text{S-OUT} \qquad \frac{}{x(z).P \xrightarrow{true,x(z)} P}\ \text{S-INP} \qquad \frac{}{\tau.P \xrightarrow{true,\tau} P}\ \text{S-TAU}$$

$$\frac{\pi.P \xrightarrow{\phi,\alpha} P'}{[x=y]\pi.P \xrightarrow{\phi\wedge[x=y],\alpha} P'}\ \text{S-MAT}$$

$$\frac{P \xrightarrow{\phi,\alpha} P'}{P \mid Q \xrightarrow{\phi,\alpha} P' \mid Q}\ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset\ \text{S-PAR-L}$$

$$\frac{P \xrightarrow{\phi,\alpha} P'}{Q \mid P \xrightarrow{\phi,\alpha} Q \mid P'}\ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset\ \text{S-PAR-R}$$

$$\frac{P \xrightarrow{\phi,\alpha} P'}{P + Q \xrightarrow{\phi,\alpha} P' + Q}\ \text{S-SUM-L} \qquad \frac{P \xrightarrow{\phi,\alpha} P'}{Q + P \xrightarrow{\phi,\alpha} Q + P'}\ \text{S-SUM-R}$$

$$\frac{P \xrightarrow{\phi,\overline{x}y} P' \qquad Q \xrightarrow{\psi,w(z)} Q'}{P \mid Q \xrightarrow{\phi\wedge\psi\wedge[x=w],\tau} P' \mid Q'\{y/z\}}\ \text{S-COMM-L}$$

$$\frac{P \xrightarrow{\phi,w(z)} P' \qquad Q \xrightarrow{\psi,\overline{x}y} Q'}{P \mid Q \xrightarrow{\phi\wedge\psi\wedge[x=w],\tau} P'\{y/z\} \mid Q'}\ \text{S-COMM-R}$$

$$\frac{P \xrightarrow{\phi,\overline{x}(z)} P' \qquad Q \xrightarrow{\psi,w(z)} Q'}{P \mid Q \xrightarrow{\phi\wedge\psi\wedge[x=z],\tau} (\nu z)(P' \mid Q')}\ \text{S-CLOSE-L}$$

$$\frac{P \xrightarrow{\phi,w(z)} P' \qquad Q \xrightarrow{\psi,\overline{x}(z)} Q'}{P \mid Q \xrightarrow{\phi\wedge\psi\wedge[w=x],\tau} (\nu z)(P' \mid Q')}\ \text{S-CLOSE-R}$$

$$\frac{P \xrightarrow{\phi,\alpha} P'}{(\nu z)P \xrightarrow{\mathcal{R}_z(\phi),\alpha} (\nu z)P'}\ z \notin \mathtt{n}(\alpha)\ \text{S-RES} \qquad \frac{P \xrightarrow{\phi,\overline{x}z} P'}{(\nu z)P \xrightarrow{\mathcal{R}_z(\phi),\overline{x}(z)} P'}\ z \neq x\ \text{S-OPEN}$$

$$\frac{P \xrightarrow{\phi,\alpha} P'}{!P \xrightarrow{\phi,\alpha} P' \mid !P}\ \text{S-REP-ACT}$$

$$\frac{P \xrightarrow{\phi,\overline{x}y} P' \qquad P \xrightarrow{\psi,w(z)} P''}{!P \xrightarrow{\phi\wedge\psi\wedge[x=w],\tau} (P' \mid P''\{y/z\}) \mid !P}\ \text{S-REP-COMM}$$

$$\frac{P \xrightarrow{\phi,\overline{x}(z)} P' \qquad P \xrightarrow{\psi,w(z)} P''}{!P \xrightarrow{\phi\wedge\psi\wedge[x=w],\tau} ((\nu z)(P' \mid P'')) \mid !P}\ \text{S-REP-CLOSE}$$

Figure B.5: The Symbolic Transition Rules

Appendix C

Compacting Parametric Traces with Type

Although the number of parametric traces generated by the parametric model for a security protocol in bounded sessions is finite, it is still too large to be handled, with lots of redundant parametric traces. For example, considering the following parametric transitions,

$$\begin{array}{c}\langle \epsilon, a(x).case\ x\ of\ \{y\}_{\mathtt{k}[A,B]}\ in\ P'\rangle \longrightarrow_p \\ \langle a(x), case\ x\ of\ \{y\}_{\mathtt{k}[A,B]}\ in\ P'\rangle \longrightarrow_p \\ \langle a(\{y\}_{\mathtt{k}[A,B]}), P'\{\{y\}_{\mathtt{k}[A,B]}/x\}\rangle\end{array}$$

There are two parametric traces generated currently, $a(x)$ and $a(\{y\}_{\mathtt{k}[A,B]})$. Actually, $a(x)$ is a redundant parametric trace, since in reality, an input action and its following validating actions, such as decryption, splitting, matching, can be regarded an atomic action, among which no attacks can occur. In the above example, it is sufficient to only check whether $a(\{y\}_{\mathtt{k}[A,B]})$ satisfies a given action term.

To reduce the redundant parametric traces, a statical analysis on a process that describes a security protocol is proposed, gathering the information of each input variable by a *type system* [96, 97], then translating the process to its corresponding *parametric process* according to the type. By a simple semantics, the parametric process generates parametric traces. The set of parametric traces generated by this semantics is a proper subset of parametric traces generated by the parametric model in Figure 5.3. We find a corresponding concrete model with type restriction, which is sound and complete with respective to the representation. In the above example, its corresponding parametric process will be

$$a(\{y\}_{\mathtt{k}[A,B]}).case\ \{y\}_{\mathtt{k}[A,B]}\ of\ \{y\}_{\mathtt{k}[A,B]}\ in\ P'\{\{y\}_{\mathtt{k}[A,B]}/x\}$$

which only generates one parametric trace, $a(\{y\}_{\mathtt{k}[A,B]})$.

C.1 Type System

Definition C.1 (Type)**.** Let $\mathcal{T}$ be the set of types. Its syntax is inductively defined as follows:

$\tau ::=$

α	Type variable
b	Base type
$\tau * \tau$	Pair type
$\sigma[\tau_1, \ldots, \tau_n]$	Binder type
$\ominus\tau \mid \ominus_+\tau \mid \ominus_-\tau \mid \ominus_?\tau$	Encrypted type
$unit$	Nil type
$\tau + \tau$	Disjoint type
$\tau \rightarrow \tau$	Arrow type

Intuitively interpreting,

- α ranges over a countable set of type variables.
- b ranges over the set of base types, which consists of an identity type i for names of principals, a nonce type n for nonces, h for hash message, and other kinds of base types, for instance, int, $char$, etc.
- The pair type $\tau * \tau$ is given to a pair message.
- The binder type $\sigma[\tau_1, \ldots, \tau_n]$ is given to a binder $\mathtt{m}[pr_1, \ldots, pr_n]$, where σ ranges over the set of binder name types, and $\tau_1, \ldots, \tau_n$ are the types of the binder's parameters $pr_1, \ldots, pr_n$, respectively. For example $k, k_+, k_-, \ldots$ are binder name types for binder names $\mathtt{k}, +\mathtt{k}, -\mathtt{k}$, respectively, so the type of a binder $\mathtt{k}[A, B]$ is $k[i, i]$. Since given a binder name type, its parameters' types will be fixed, for simplicity, we usually use a binder name type to represent a binder's type. For instance, the type of $\mathtt{k}[A, B]$ is usually abbreviated to k.
- The shared-key encryption type, $\ominus\tau$, is given to an encrypted message encrypted by a shared-key, and τ is the type for its plain message. Similarly, $\ominus_+\tau$ is for a public-key encrypted message, and $\ominus_-\tau$ is for a private-key encrypted message, say, a digital signature. $\ominus_?\tau$ is for an encrypted message whose key cannot be decided statically.
- The type $unit$ is a nil type for the $\mathbf{0}$ process.
- The disjoint type $\tau + \tau$ is given to a composition process, $P \| Q$.
- The arrow type $\tau \rightarrow \tau$ is given to an input process, which is similar to the type of the abstraction in λ-calculus.

We use an expression e where $e \in \mathcal{M} \cup \mathcal{P}$ to describe a message or a process. Let Γ be a type environment mapping from the set of variables $\mathcal{V}$, to the set of types $\mathcal{T}$. The typing inference system has the form $\Gamma \vdash_t e : \tau$, in which Γ is a type environment, e is the expression whose type will be inferred and τ is the type of e. If the type environment is an empty set, the form is abbreviated to $\vdash_t e : \tau$. Furthermore, we presuppose a function $\mathtt{TypeOf} : \mathscr{P}(\mathcal{N}) \cup \mathscr{P}(\mathcal{B}) \rightarrow \mathcal{T}$ that assigns a type to a set of names or binder names. Here

$$\frac{}{\Gamma \vdash_t x : \tau}\ (x,\tau) \in \Gamma \quad \text{Msg\_Variable} \qquad \frac{}{\Gamma \vdash_t n : b}\ b = \texttt{TypeOf}(n) \quad \text{Msg\_Name}$$

$$\frac{}{\Gamma \vdash_t \mathcal{H}(M) : b}\quad b = \texttt{TypeOf}(\mathcal{H}(M)) \quad \text{Msg\_Hash}$$

$$\frac{\Gamma \vdash_t \tilde{pr} : \tau_1 * \ldots * \tau_n}{\Gamma \vdash_t \mathtt{m}[\tilde{pr}] : \sigma[\tau_1, \ldots, \tau_n]}\ \sigma = \texttt{TypeOf}(\mathtt{m}) \quad \text{Msg\_Binder}$$

$$\frac{\Gamma \vdash_t M : \tau_1 \quad \Gamma \vdash_t N : \tau_2}{\Gamma \vdash_t (M,N) : \tau_1 * \tau_2} \quad \text{Msg\_Pair}$$

$$\frac{\Gamma \vdash_t M : \tau \quad \Gamma \vdash_t L : k^*}{\Gamma \vdash_t \{M\}_L : \ominus^*\tau}\quad \ominus^*\tau = \begin{cases} \ominus\tau & k^* = k \\ \ominus_+\tau & k^* = k_+ \\ \ominus_-\tau & k^* = k_- \\ \ominus_?\tau & k^* = \alpha \end{cases} \quad \text{Msg\_Enc}$$

$$\frac{}{\Gamma \vdash_t \mathbf{0} : unit}\ \text{Nil} \qquad \frac{\Gamma \vdash_t P : \tau_2 \quad \Gamma \vdash_t n : \tau_1}{\Gamma \vdash_t (\nu\, n)P : \tau_2}\ \text{Restriction}$$

$$\frac{\Gamma, \{x : \tau_1\} \vdash_t P : \tau_2}{\Gamma \vdash_t (\texttt{new}\, x : \mathcal{A})P : \tau_2}\ \tau_1 = \texttt{TypeOf}(\mathcal{A}) \quad \text{New}$$

$$\frac{\Gamma, \{x : \tau_1\} \vdash_t P : \tau_2}{\Gamma \vdash_t a(x).P : \tau_1 \to \tau_2}\ \text{Input} \qquad \frac{\Gamma \vdash_t P : \tau_1 \quad \Gamma \vdash_t M : \tau_2}{\Gamma \vdash_t \overline{a}M.P : \tau_1}\ \text{Output}$$

$$\frac{\Gamma, \{x : \tau_1, y : \tau_2\} \vdash_t P : \tau_3 \quad \Gamma \vdash_t M : \tau_1 * \tau_2}{\Gamma \vdash_t let\ (x,y) = M\ in\ P : \tau_3}\ \text{Pair}$$

$$\frac{\Gamma, \{x : \tau_1\} \vdash_t P : \tau_2 \quad \Gamma \vdash_t M : \ominus^*\tau_1 \quad \Gamma \vdash_t L : k^*}{\Gamma \vdash_t case\ M\ of\ \{x\}_L\ in\ P : \tau_2}\quad \ominus^*\tau = \begin{cases} \ominus\tau & k^* = k \\ \ominus_+\tau & k^* = k_- \\ \ominus_-\tau & k^* = k_+ \\ \ominus_?\tau & k^* = \alpha \end{cases} \quad \text{Dec}$$

$$\frac{\Gamma \vdash_t M : \tau_1 \quad \Gamma \vdash_t N : \tau_1 \quad \Gamma \vdash_t P : \tau_2}{\Gamma \vdash_t [M = N]\, P : \tau_2}\ \text{Match}$$

$$\frac{\Gamma \vdash_t P : \tau_1 \quad \Gamma \vdash_t N : \tau_2}{\Gamma \vdash_t P \| Q : \tau_1 + \tau_2}\ \text{Composition}$$

Figure C.1: Typing Rules

$\texttt{TypeOf}(n)$ is the abbreviation of $\texttt{TypeOf}(\{n\})$. The typing rules for the expressions are given in Figure C.1.

The typing system does not provide an easy way to assign a type to an expression e. Thus a type algorithm is provided and its correctness is verified. In the algorithm, given a typing environment Γ and an expression e, a substitution θ mapping from type variables to types and a type τ can be calculated, which satisfy

$$\Gamma\theta \vdash_t e : \tau$$

Before defining the algorithm, we provide a type unification algorithm, which can be used in the type inference algorithm. An occurrence check function $FTV(\tau, \alpha)$ is presupposed as usual, which satisfies that α does not occur in τ if $FTV(\tau, \alpha) = True$. The unification algorithm has the following form $\texttt{Unify}(\tau, \tau') = (\theta, \sigma)$. That is, given two types τ and τ', it either returns a substitution θ and a type σ that satisfy $\tau\theta = \tau'\theta = \sigma$, or raises failure.

The `Unify` and `Infer` algorithms are given in Algorithm 3 and Algorithm 4, respectively. We will prove that every type calculated by the algorithm `Infer` can be inferred by the typing rules in Figure C.1.

Algorithm 3 The Unification Algorithm

$\texttt{Unify}(\tau, \tau') = (\theta, \sigma)$

1. $\texttt{Unify}(\alpha, \tau') = (\{\tau'/\alpha\}, \tau')$, if $FTV(\tau', \alpha)$.
2. $\texttt{Unify}(\tau, \tau) = (Id, \tau)$.
3. let $\texttt{Unify}(\tau_1, \tau_1') = (\theta_1, \sigma_1)$, $\texttt{Unify}(\tau_2\theta_1, \tau_2'\theta_1) = (\theta_2, \sigma_2)$ in
   $\texttt{Unify}(\tau_1 * \tau_2, \tau_1' * \tau_2') = (\theta_1\theta_2, \sigma_1\theta_2 * \sigma_2)$.
4. let $\texttt{Unify}(\tau_1, \tau_1') = (\theta, \sigma_1)$, $\sigma = \ominus^*\sigma_1$ in
   $\texttt{Unify}(\ominus^*\tau_1, \ominus^*\tau_1') = (\theta, \sigma)$ $(\ominus^* \in \{\ominus_+, \ominus_-, \ominus, \ominus_?\})$.
5. let $\texttt{Unify}(\tau_1, \tau_1') = (\theta_1, \sigma_1)$, $\texttt{Unify}(\tau_2\theta_1, \tau_2'\theta_1) = (\theta_2, \sigma_2)$ in
   $\texttt{Unify}(\tau_1 + \tau_2, \tau_1' + \tau_2') = (\theta_1\theta_2, \sigma_1\theta_2 + \sigma_2)$.
6. let $\texttt{Unify}(\tau_1, \tau_1') = (\theta_1, \sigma_1)$, $\texttt{Unify}(\tau_2\theta_1, \tau_2'\theta_1) = (\theta_2, \sigma_2)$ in
   $\texttt{Unify}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') = (\theta_1\theta_2, \sigma_1\theta_2 \to \sigma_2)$.
7. raise error.

Algorithm 4 The Type Inference Algorithm

$\texttt{Infer}(\Gamma, e) = (\theta, \tau)$

1. $\texttt{Infer}(\Gamma, x) = (Id, \tau)$ where $(x, \tau) \in \Gamma$.
2. $\texttt{Infer}(\Gamma, n) = (Id, b)$ where $b = \texttt{TypeOf}(n)$.
3. $\texttt{Infer}(\Gamma, \mathcal{H}(M)) = (Id, h)$ where $h = \texttt{TypeOf}(\mathcal{H}(M))$.
4. Let $\texttt{Infer}(\Gamma, \tilde{pr}) = (\theta_1, \tau_1 * \ldots * \tau_n)$ in $\texttt{Infer}(\Gamma, \texttt{m}[\tilde{pr}]) = (\theta_1, \sigma[\tau_1, \ldots, \tau_n])$ where $\sigma = \texttt{TypeOf}(\texttt{m})$.
5. Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$, $\texttt{Infer}(\Gamma\theta_1, N) = (\theta_2, \tau_2)$ in $\texttt{Infer}(\Gamma, (M, N)) = (\theta_1\theta_2, (\tau_1\theta_2) * \tau_2)$.
6. Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$, $\texttt{Infer}(\Gamma\theta_1, L) = (\theta_2, \tau_2)$, $\texttt{Unify}(\tau_2, k^*) = (\vartheta, k^*)$ in $\texttt{Infer}(\Gamma, \{M\}_L) = (\theta_1\theta_2\vartheta, \ominus^*(\tau_1\theta_2))$ where $\ominus^*(\tau_1\theta_2) = \begin{cases} \ominus(\tau_1\theta_2) & k^* = k \\ \ominus_+(\tau_1\theta_2) & k^* = k_+ \\ \ominus_-(\tau_1\theta_2) & k^* = k_- \\ \ominus_?(\tau_1\theta_2) & k^* = \alpha \end{cases}$.
7. $\texttt{Infer}(\Gamma, \mathbf{0}) = (Id, unit)$.
8. Let $\texttt{Infer}(\Gamma \cup \{(x, \tau_1)\}, P) = (\theta, \tau_2)$ in $\texttt{Infer}(\Gamma, (\texttt{new}\, x : \mathcal{A})P) = (\theta, \tau_2)$ where $\tau_1 = \texttt{TypeOf}(\mathcal{A})$.
9. Let $\texttt{Infer}(\Gamma, P) = (\theta_1, \tau_1)$, $\texttt{Infer}(\Gamma, n) = (\theta_2, \tau_2)$ in $\texttt{Infer}(\Gamma, (\nu\, n)P) = (\theta_1\theta_2, \tau_1)$ where $\tau_2 = \texttt{TypeOf}(n)$.
10. Let $\texttt{Infer}(\Gamma \cup \{(x, \alpha)\}, P) = (\theta, \tau)$ in $\texttt{Infer}(\Gamma, a(x).P) = (\theta, \alpha\theta \to \tau)$, where $\forall\tau.(x', \tau) \in \Gamma$, $FTV(\tau, \alpha) = True$.
11. Let $\texttt{Infer}(\Gamma, P) = (\theta, \tau)$ in $\texttt{Infer}(\Gamma, \overline{a}M.P) = (\theta, \tau)$.
12. Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$, $\texttt{Unify}(\alpha * \beta, \tau_1) = (\vartheta, \varrho)$, $\texttt{Infer}(\Gamma\theta_1\vartheta \cup \{(x, \alpha), (y, \beta)\}\vartheta, P) = (\theta_2, \tau_2)$ in $\texttt{Infer}(\Gamma, let\ (x, y) = M\ in\ P) = (\theta_1\vartheta\theta_2, \tau_2)$ where $\forall\tau.(x', \tau) \in \Gamma$, $FTV(\tau, \alpha) = FTV(\tau, \beta) = True$.
13. Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$, $\texttt{Infer}(\Gamma\theta_1, L) = (\theta_2, \tau_2)$, $\texttt{Unify}(\ominus^*\alpha, \tau_1\theta_2) = (\vartheta, \varrho)$, $\texttt{Infer}(\Gamma\theta_1\theta_2\vartheta \cup \{(x, \alpha)\}\vartheta, P) = (\theta_3, \tau_3)$ in $\texttt{Infer}(\Gamma, case\ M\ of\ \{x\}_L\ in\ P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$ where $\forall\tau.(x', \tau) \in \Gamma$, $FTV(\tau, \alpha) = True$, and $\ominus^*\alpha = \begin{cases} \ominus(\tau_1\theta_2) & \tau_2 = k \\ \ominus_+(\tau_1\theta_2) & \tau_2 = k_- \\ \ominus_-(\tau_1\theta_2) & \tau_2 = k_+ \\ \ominus_?(\tau_1\theta_2) & \tau_2 = \beta \end{cases}$.
14. Let $\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1)$, $\texttt{Infer}(\Gamma\theta_1, N) = (\theta_2, \tau_2)$, $\texttt{Unify}(\tau_1\theta_2, \tau_2) = (\vartheta, \varrho)$ $\texttt{Infer}(\Gamma\theta_1\theta_2\varrho, P) = (\theta_3, \tau_3)$ in $\texttt{Infer}(\Gamma, [M = N]P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$.
15. Let $\texttt{Infer}(\Gamma, P) = (\theta_1, \tau_1)$, $\texttt{Infer}(\Gamma\theta_1, Q) = (\theta_2, \tau_2)$ in $\texttt{Infer}(\Gamma, P \| Q) = (\theta_1\theta_2, \tau_1\theta_2 + \tau_2)$.

Lemma C.1. *Let e be an expression, τ be a type, Γ be a type environment and θ be a substitution mapping from type variables to types. If $\Gamma \vdash_t e : \tau$, then $\Gamma\theta \vdash_t e : \tau\theta$.*

Proof. Applying structural induction to the expression e, we only show the base cases and two inductive steps here; the remaining cases are quite similar.

1. Case $e = x$: Because $\Gamma \vdash_t x : \tau$, we have $(x, \tau) \in \Gamma$. Let $\Gamma_1 = \Gamma \backslash \{(x, \tau)\}$, so $\Gamma\theta = \Gamma_1\theta \cup \{(x, \tau)\}\theta = \Gamma_1\theta \cup \{(x, \tau\theta)\}$, and thus $\Gamma\theta \vdash_t x : \tau\theta$ according to typing rule Msg_Variable.

2. Case $e = n$: Because $\Gamma \vdash_t n : \tau$, we have $\tau = \texttt{TypeOf}(n)$, and $\tau\theta = \tau$ for every substitution θ. By typing rule Msg_Name, $\Gamma\theta \vdash_t n : \texttt{TypeOf}(n)$.

3. Case $e = (M, N)$: By typing rules, $\Gamma \vdash_t M : \tau_1$ and $\Gamma \vdash_t N : \tau_2$ are satisfied. Given a substitution θ, we have $\Gamma\theta \vdash_t M : \tau_1\theta$ and $\Gamma\theta \vdash_t M : \tau_2\theta$ according to induction hypothesis. So $\Gamma\theta \vdash_t (M, N) : (\tau_1 * \tau_2)\theta$.

4. Case $e = a(x).P$: By induction hypothesis, $\Gamma\theta \cup \{x, \tau_1\}\theta = \Gamma\theta \cup \{x, \tau_1\theta\} \vdash_t P : \tau_2\theta$ is satisfied. Thus $\Gamma\theta \vdash_t a(x)P : (\tau_1 \to \tau_2)\theta$.

□

Theorem C.2 (Soundness of type inference)**.** *Let e be an expression, Γ be a typing environment and θ be a substitution. If $\texttt{Infer}(\Gamma, e) = (\theta, \tau)$, $\Gamma\theta \vdash_t e : \tau$ can be inferred.*

Proof. Applying structural induction to the expression e. one base step and three inductive steps are proposed; the remaining cases are simple and similar. We use the same notations as in the algorithm.

1. Case $e = x$: We have $\texttt{Infer}(\Gamma, x) = (Id, \tau)$, because $\Gamma Id = \Gamma$ and $(x, \tau) \in \Gamma$, $\Gamma \vdash_t x : \tau$ is inferred by the typing rule Msg_Variable.

2. Case $e = (M, N)$: by induction hypothesis, we have two derivations:

$$\texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1) \quad \mapsto \quad \Gamma\theta_1 \vdash_t M : \tau_1 \tag{C.1}$$

$$\texttt{Infer}(\Gamma\theta_1, N) = (\theta_2, \tau_2) \quad \mapsto \quad \Gamma\theta_1\theta_2 \vdash_t N : \tau_2 \tag{C.2}$$

   By applying Lemma C.1 to (C.1), we have

$$\Gamma\theta_1\theta_2 \vdash_t M : \tau_1\theta_2 \tag{C.3}$$

   By applying typing rule Msg_Pair to (C.2) and (C.3), we have

$$\Gamma\theta_1\theta_2 \vdash_t (M, N) : \tau_1\theta_2 * \tau_2$$

   which is the expected result of

$$\texttt{Infer}(\Gamma, (M, N)) = (\theta_1\theta_2, (\tau_1\theta_2) * \tau_2)$$

3. Case $e = a(x).P$: By induction hypothesis, we get a derivation:

$$\texttt{Infer}(\Gamma \cup \{(x, \alpha)\}, P) = (\theta, \tau) \quad \mapsto \quad \Gamma\theta \cup \{(x, \alpha)\}\theta \vdash_t P : \tau \tag{C.4}$$

   By applying typing rule Input to (C.4), we have

$$\Gamma\theta \vdash_t a(x).P : \alpha\theta \to \tau$$

   which is the result of $\texttt{Infer}(\Gamma, a(x).P) = (\theta, \alpha\theta \to \tau)$.

4. Case $e = case\ M\ of\ \{x\}_L\ in\ P$: By the induction hypothesis, we have two derivations and a unification:

$$\begin{aligned} \texttt{Infer}(\Gamma, M) = (\theta_1, \tau_1) &\mapsto \Gamma\theta_1 \vdash_t M : \tau_1 && \text{(C.5)} \\ \texttt{Infer}(\Gamma\theta_1, L) = (\theta_2, \tau_2) &\mapsto \Gamma\theta_1\theta_2 \vdash_t L : \tau_2 && \text{(C.6)} \\ \texttt{Unify}(\ominus^*\alpha, \tau_1\theta_2) = (\vartheta, \varrho) &\mapsto \ominus^*(\alpha\vartheta) = \tau_1\theta_2\vartheta = \varrho && \text{(C.7)} \end{aligned}$$

$$\texttt{Infer}(\Gamma\theta_1\theta_2\vartheta \cup \{(x,\alpha)\}\theta_2\vartheta, P) = (\theta_3, \tau_3) \mapsto \Gamma\theta_1\theta_2\vartheta\theta_3 \cup \{(x,\alpha)\vartheta\theta_3\} \vdash_t P : \tau_3 \quad \text{(C.8)}$$

By applying Lemma C.1 to (C.5), we have

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash_t M : \tau_1\theta_2\vartheta\theta_3 \quad \text{(C.9)}$$

Note that α does not occur in θ_1 and θ_2, and thus $\{(x,\alpha)\}\vartheta\theta_3 = \{(x,\alpha)\}\theta_1\theta_2\vartheta\theta_3$. So after unification, the type of x is $\alpha\vartheta\theta_2$, and the type of $\{x\}_L$ is $\ominus^*(\alpha\vartheta\theta_2)$. And because of (C.7), we have

$$\ominus^*(\alpha\vartheta\theta_3) = \tau_1\theta_2\vartheta\theta_3$$

and thus

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash_t M : \ominus^*(\alpha\vartheta\theta_3)$$

Furthermore, because of (C.7),

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash_t L : \tau_2\vartheta_2\theta_3$$

So, using typing rule *Dec*, we have

$$\Gamma\theta_1\theta_2\vartheta\theta_3 \vdash_t case\ M\ of\ \{x\}_L\ in\ P : \tau_3$$

which is the result of

$$\texttt{Infer}(\Gamma, case\ M\ of\ \{x\}_L\ in\ P) = (\theta_1\theta_2\vartheta\theta_3, \tau_3)$$

□

C.2 Translating to Parametric Processes

A set of *parametric processes* is defined as follows,

Definition C.2 (Parametric Processes)**.** Let $\hat{\mathcal{P}}$ be a countable set of parametric processes, which is indicated by $\hat{P}, \hat{Q}, \hat{R}, \ldots$. The syntax of processes is defined as follows,

$$\begin{aligned} \hat{P} ::= \ & \mathbf{0} \mid \overline{a}M.\hat{P} \mid a(M).\hat{P} \mid [M = N]\,\hat{P} \mid (\texttt{new}\ x : \mathcal{A})\hat{P} \mid (\nu\, n)\hat{P} \mid let\ (M,N) = L\ in\ \hat{P} \\ & \mid case\ M\ of\ \{N\}_L\ in\ \hat{P} \mid \hat{P} \| \hat{Q} \end{aligned}$$

where M, N, L are messages defined in Definition 2.1.

Given a closed process P, we try to mark each sub-expression whose type is a type variable with a fresh variable. Thus P can be translated into a parametric process $\hat{P}$. For this purpose, given a closed process P and its type τ, an inference system is proposed, which infers a substitution θ mapping from $\mathcal{V}$ to $\mathcal{M}$. The inference system has the form $\Delta \vdash_p P : \tau \Rightarrow \theta$, in which Δ is a context environment mapping from $\mathcal{V}$ to $\mathcal{M}$, P is a closed process and τ is its type. The corresponding parametric process $\hat{P}$ is obtained by applying the substitution θ to the process P. We name $\hat{P}$ the *abstraction* of P, and P the *concretization* of $\hat{P}$.

In the inference system, some functions are predefined. $\texttt{FreVar} : \Delta \to \mathcal{V}$ generates a fresh variable that does not occur in $\texttt{Dom}(\Delta)$. $\texttt{Binder} : \mathcal{T} \to \mathcal{B}$ obtains the corresponding binder name from a binder type. The inference system is given in Figure C.2.

Example C.1. By the inference system, the process of the Abadi-Gordon protocol for authentication, SYS_a^{AG} described in Subsection 5.3.1 will be translated into the following parametric processes:

$$\begin{aligned}
\hat{A}_p \triangleq & (\texttt{new}\ x_1 : \mathcal{I})(\nu\, M)\overline{a1}(A, \{x_1, \texttt{k}[A, x_1]\}_{\texttt{k}[A,S]}).\overline{a2}(A, \{A, M\}_{\texttt{k}[A,x_1]}).\mathbf{0} \\
\hat{B}_p \triangleq & b1(\{A, z_1\}_{\texttt{k}[B,S]}).\ case\ \{A, z_1\}_{\texttt{k}[B,S]}\ of\ \{A, z_1\}_{\texttt{k}[B,S]}\ in\ let\ (A, z_1) = \\
& (A, z_1)\ in\ [A = A]\, b2(A, \{A, w_1''\}_{t_1}).\ let\ (A, (A, \{A, w_1''\}_{t_1})) = \\
& (A, (A, \{A, w_1''\}_{t_1}))\ in\ [A = A]\ case\ \{A, w_1''\}_{t_1}\ of\ \{A, w_1''\}_{z_1} \\
& in\ let\ (A, w_1'') = (A, w_1'')\ in\ [A = A]\, \overline{acc}\, (A, \{A, w_1''\}_{t_1}).\mathbf{0} \\
\hat{S}_p \triangleq & s1(x, \{y, z\}_{\texttt{k}[x_k,S]}).let\ (x, \{y, z\}_{\texttt{k}[x_k,S]}) = (x, \{y, z\}_{\texttt{k}[x_k,S]})\ in\ case\ \{y, z\}_{\texttt{k}[x_k,S]} \\
& of\ \{y, z\}_{\texttt{k}[x,S]}\ in\ let\ (y, z) = (y, z)\ in\ \overline{s2}\{x, z\}_{\texttt{k}[y,S]}.\mathbf{0} \\
\hat{SYS}_p^{AG} \triangleq & A_p \| S_p \| B_p
\end{aligned}$$

From the example introduced above, we can see that except input and output actions, other actions, such as splitting, decrypting, and match, are all redundant action essentially. So $\hat{B}_p$ can be simplified as

$$\hat{B}_p \triangleq b1(\{A, z_1\}_{\texttt{k}[B,S]}).b2(A, \{A, w_1''\}_{t_1}).\overline{acc}\, (A, \{A, w_1''\}_{t_1}).\mathbf{0}$$

Example C.2. The process of the NSPK protocol for authentication, SYS_a^{NSPK} described in Subsection 5.3.2 will be translated into the following parametric processes.

$$\begin{aligned}
\hat{A}_p \triangleq & (\texttt{new}\ x_a : \mathcal{I})(\nu\, N_A)\overline{a1}\{A, N_A\}_{+\texttt{k}[x_a]}.a2(\{z_a, z_a'\}_{-\texttt{k}[x_k]}). \\
& case\ \{z_a, z_a'\}_{-\texttt{k}[x_k]}\ of\ \{z_a, z_a'\}_{-\texttt{k}[A]}\ in\ let\ (z_a, z_a') = (z_a, z_a')\ in \\
& [z_a = N_A]\overline{a3}\{z_a'\}_{+\texttt{k}[x_a]}.\mathbf{0} \\
\hat{B}_p \triangleq & (\nu\, N_B)b1(\{x_b, y_b\}_{-\texttt{k}[y_k]}).\ case\ \{x_b, y_b\}_{-\texttt{k}[y_k]}\ of\ \{x_b, y_b\}_{-\texttt{k}[B]}\ in \\
& let\ (x_b, y_b) = (x_b, y_b)\ in\ [x_b = A]\, \overline{b2}\{y_b, N_B\}_{+\texttt{k}[A]}.b3(\{z_b\}_{-\texttt{k}[y_k]}). \\
& case\ \{z_b\}_{-\texttt{k}[y_k]}\ of\ \{z_b\}_{-\texttt{k}[B]}\ in\ [z_b = N_B]\, \overline{acc}\, \{z_b\}_{-\texttt{k}[y_k]}.\mathbf{0} \\
\hat{SYS}_p^{NSPK} \triangleq & A_p \| B_p
\end{aligned}$$

$$\frac{}{\Delta \vdash_p x : \alpha \Rightarrow \{x_1/x\}} \; x_1 = \texttt{FreVar}(\Delta) \;\; \text{Type\_variable}$$

$$\frac{}{\Delta \vdash_p x : b \Rightarrow \{x_b/x\}} \; x_b = \texttt{FreVar}(\Delta) \;\; \text{Base\_type}$$

$$\frac{\Delta \vdash_p x' : \tau_1 * \ldots * \tau_n \Rightarrow \theta}{\Delta \vdash_p x : \sigma[\tau_1, \ldots, \tau_n] \Rightarrow \{\texttt{m}[\theta(x')]/x\}} \; \texttt{m} = \texttt{Binder}(\sigma), x' = \texttt{FreVar}(\Delta) \;\; \text{Binder\_type}$$

$$\frac{\Delta \vdash_p x' : \tau_1 \Rightarrow \theta_1 \quad \Delta \vdash_p x'' : \tau_2 \Rightarrow \theta_2}{\Delta \vdash_p x : \tau_1 * \tau_2 \Rightarrow \{(\theta_1(x'), \theta_2(x''))/x\}} \; x', x'' = \texttt{FreVar}(\Delta) \; \text{Pair\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus\tau \Rightarrow \{\{\theta(x')\}_{\texttt{k}[x_a, x_b]}/x\}} \; x_a, x_b, x' = \texttt{FreVar}(\Delta) \;\; \text{Sencryption\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus_+\tau \Rightarrow \{\{\theta(x')\}_{+\texttt{k}[x_a]}/x\}} \; x_a, x' = \texttt{FreVar}(\Delta) \;\; \text{Pencryption\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus_-\tau \Rightarrow \{\{\theta(x')\}_{-\texttt{k}[x_a]}/x\}} \; x_a, x' = \texttt{FreVar}(\Delta) \;\; \text{Signature\_type}$$

$$\frac{\Delta \vdash_p x' : \tau \Rightarrow \theta}{\Delta \vdash_p x : \ominus_?\tau \Rightarrow \{\{\theta(x')\}_{x_a}/x\}} \; x_a, x' = \texttt{FreVar}(\Delta) \;\; \text{Gencryption\_type}$$

$$\frac{}{\Delta \vdash_p \mathbf{0} : \texttt{unit} \Rightarrow \{\}} \; \text{Nil} \qquad \frac{\Delta \vdash_p x : \tau_1 \Rightarrow \theta_1 \quad \Delta, \theta_1 \vdash_p P : \tau_2 \Rightarrow \theta_2}{\Delta \vdash_p a(x).P : \tau_1 \rightarrow \tau_2 \Rightarrow \theta_1 \cup \theta_2} \; \text{Input}$$

$$\frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p \overline{a}M.P : \tau \Rightarrow \theta} \; \text{Output}$$

$$\frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p (\texttt{new}\, x : \mathcal{A})\, P : \tau \Rightarrow \theta \cup \{x_1/x\}} \; x_1 = \texttt{FreVar}(\Delta) \;\; \text{New}$$

$$\frac{\Delta, \{M_1/x, M_2/y\} \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p \mathit{let}\ (x, y) = M\ \mathit{in}\ P : \tau \Rightarrow \theta \cup \{M_1/x, M_2/y\}} \; (\Delta(M) = (M_1, M_2))\text{Pair}$$

$$\frac{\Delta, \{M/x\} \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p \mathit{case}\ M\ \mathit{of}\{x\}_L\ \mathit{in}\ P : \tau \Rightarrow \theta \cup \{M/x\}} \; (\Delta(M) = \{M\}_L)\ \text{Dec}$$

$$\frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p [M = N]P : \tau \Rightarrow \theta} \; \text{Match} \qquad \frac{\Delta \vdash_p P : \tau \Rightarrow \theta}{\Delta \vdash_p (\nu\, n).P : \tau \Rightarrow \theta} \; \text{Restriction}$$

$$\frac{\Delta \vdash_p P : \tau_1 \Rightarrow \theta_1 \quad \Delta \vdash_p Q : \tau_2 \Rightarrow \theta_2}{\Delta \vdash_p P \| Q : \tau_1 + \tau_2 \Rightarrow \theta_1 \cup \theta_2} \; \text{Composition}$$

Figure C.2: Inference Rules for Parametric Processes

Similarly, we define an configuration is a pair $\langle \hat{s}, \hat{P} \rangle$, where $\hat{s}$ is a parametric trace and $\hat{P}$ is a parametric process. The transition relation of parametric processes is given by the rules in Figure C.3. Note that $TPDEC$, $TPAIR$, $TPMATCH$ are only facilities for the later proof. They will not be adopted in the implementation.

$$
\begin{array}{rl}
(TPINPUT) & \langle \hat{s}, a(M).\hat{P} \rangle \longrightarrow_p^t \langle \hat{s}.a(M), \hat{P} \rangle \\
(TPOUTPUT) & \langle \hat{s}, \overline{a}M.\hat{P} \rangle \longrightarrow_p^t \langle \hat{s}.\overline{a}M, \hat{P} \rangle \\
(TPDEC) & \langle \hat{s}, case\ \{M\}_L\ of\ \{M\}_{\mathtt{Opp}(L)}\ in\ \hat{P} \rangle \longrightarrow_p^t \langle \hat{s}, \hat{P} \rangle \\
(TPPAIR) & \langle \hat{s}, let\ (M,N) = (M,N)\ in\ \hat{P} \rangle \longrightarrow_p^t \langle \hat{s}, \hat{P} \rangle \\
(TPNEW) & \langle \hat{s}, (\mathtt{new}\, x : \mathcal{A})\hat{P} \rangle \longrightarrow_p^t \langle \hat{s}, \hat{P} \rangle \\
(TPRESTRICTION) & \langle \hat{s}, (\nu\, n)P \rangle \longrightarrow_p^t \langle \hat{s}, P\{m/n\} \rangle \ \ m = \mathtt{freshN}(V) \\
(TPMATCH) & \langle \hat{s}, [M = M]\hat{P} \rangle \longrightarrow_p^t \langle \hat{s}, \hat{P} \rangle \\
(TPLCOM) & \dfrac{\langle \hat{s}, \hat{P} \rangle \longrightarrow_p^t \langle \hat{s}', \hat{P}' \rangle}{\langle \hat{s}, \hat{P} \| \hat{Q} \rangle \longrightarrow_p^t \langle \hat{s}', \hat{P}' \| \hat{Q} \rangle} \\
(TPRCOM) & \dfrac{\langle \hat{s}, \hat{Q} \rangle \longrightarrow_p^t \langle \hat{s}', \hat{Q}' \rangle}{\langle \hat{s}, \hat{P} \| \hat{Q} \rangle \longrightarrow_p^t \langle \hat{s}', \hat{P} \| \hat{Q}' \rangle} \\
(TPSTR) & \dfrac{\hat{P} \equiv \hat{P}' \quad \langle s, \hat{P}' \rangle \longrightarrow_p^t \langle s', \hat{Q}' \rangle \quad \hat{Q}' \equiv \hat{Q}}{\langle s, \hat{P} \rangle \longrightarrow_p^t \langle s', \hat{Q} \rangle}
\end{array}
$$

Figure C.3: Parametric Transition Rules for Parametric Processes

The set of parametric traces generated by transitions rules in Figure C.3 is obvious a proper subset of the set of parametric traces generated by transitions rules in Figure 5.3. From Theorem 4.1, we know that a parametric model has the same representative ability as its corresponding concrete model. Thus, parametric traces generated by transitions rules in Figure C.3 have the less representative ability than its traces generated by transitions rules in Figure 5.1.

To find a sound and complete concrete model, we only modify the $INPUT$ rule in the concrete transition rules in Figure 5.1 with a type constraint. That is, a message M deduced by the s in the configuration $\langle s, a(x).P \rangle$, whose type can be unified with the type of x, will be instantiated to x. The concrete transition rules with the type constraint are given in Figure C.4. The new concrete model then has the same representative ability as its corresponding parametric model in Figure C.3.

Theorem C.3 (Soundness and completeness)**.** *Let $\langle \epsilon, P \rangle$ be an initial configuration, let s' be a trace, and let $\hat{P}$ be the abstraction of P. $\langle \epsilon, P \rangle \longrightarrow^{t*} \langle s', P' \rangle$ for some P', if and only if there exists $\hat{s}'$, satisfying $\langle \epsilon, \hat{P} \rangle \longrightarrow_p^{t*} \langle \hat{s}', \hat{P}' \rangle$ for some $\hat{P}'$, and s' is a concretization of $\hat{s}'$.*

Proof. We only show the $TINPUT$ and $TPINPUT$ cases in this proof, other cases are similar to those in proof of Theorem 4.1. A detailed proof can be found in [78].

"⇒": By an induction on the number of transitions $\longrightarrow^t$ and $\longrightarrow_p^t$, the proof is trivial in the zero-step. We assume in the n-th step the property holds. That is, for each trace s gained in the n-th $\longrightarrow^t$ step, there exists an $\hat{s}$ obtained by the n-th $\longrightarrow_p^t$ step, and $\hat{s}\vartheta = s$

$$(TINPUT)\quad \langle s, a(x).P : \tau_1 \to \tau_2\rangle \longrightarrow^t \langle s.a(M), P\{M/x\}\rangle \quad s \vdash M,\quad \vdash_t M : \tau_1$$

$$(TOUTPUT)\quad \langle s, \overline{a}M.P\rangle \longrightarrow^t \langle s.\overline{a}M, P\rangle$$

$$(TDEC)\quad \langle s, case\ \{M\}_L\ of\ \{x\}_{L'}\ in\ P\rangle \longrightarrow^t \langle s, P\{M/x\}\rangle \quad L' = \mathtt{Opp}(L)$$

$$(TPAIR)\quad \langle s, let\ (x,y) = (M,N)\ in\ P\rangle \longrightarrow^t \langle s, P\{M/x, N/y\}\rangle$$

$$(TNEW)\quad \langle s, (\mathtt{new}\ x : \mathcal{A})P\rangle \longrightarrow^t \langle s, P\{m/x\}\rangle \quad n \in \mathcal{A}$$

$$(TRESTRICTION)\quad \langle s, (\nu\, n)P\rangle \longrightarrow^t \langle s, P\{m/n\}\rangle \quad m = \mathtt{freshN}(V)$$

$$(TMATCH)\quad \langle s, [M = M]P\rangle \longrightarrow^t \langle s, P\rangle$$

$$(TLCOM)\quad \frac{\langle s, P\rangle \longrightarrow^t \langle s', P'\rangle}{\langle s, P\|Q\rangle \longrightarrow^t \langle s', P'\|Q\rangle}$$

$$(TRCOM)\quad \frac{\langle s, Q\rangle \longrightarrow^t \langle s', Q'\rangle}{\langle s, P\|Q\rangle \longrightarrow^t \langle s', P\|Q'\rangle}$$

$$(TSTR)\quad \frac{P \equiv P' \quad \langle s, P'\rangle \longrightarrow^t \langle s', Q'\rangle \quad Q' \equiv Q}{\langle s, P\rangle \longrightarrow^t \langle s', Q\rangle}$$

Figure C.4: Concrete Transition Rules with Type Constraint

holds for some substitution ϑ from parametric variables to ground messages. Now, we perform a case analysis on the $n+1$ step.

Case $\langle s, a(x).P\rangle$: If $\langle s, a(x).P\rangle \longrightarrow^t \langle s.a(M), P\{M/x\}\rangle$, then the type of the ground message M can be unified with the type τ of x. So after applying the inference system in Figure C.1 to the process, x in the corresponding parametric process will be substituted to a message M' that can be unified by M. Let $\vartheta' = \mathtt{Uni}(M, M')$, and $\hat{P}$ be the abstraction of P, then we have $\langle \hat{s}, a(M').\hat{P}\rangle \longrightarrow_p^t \langle \hat{s}.a(M'), \hat{P}\rangle$ and $s.a(M) = \hat{s}.a(M')(\vartheta \cup \vartheta')$.

"$\Leftarrow$": By an induction on the number of transitions $\longrightarrow_p^t$ and $\longrightarrow^t$, the proof is trivial in the zero-step. We assume in the n-th step the property holds, that is, for each parametric trace $\hat{s}$ gained by the n-th $\longrightarrow_p^t$ step, if there exists a substitution ϑ from parametric variables to ground messages, and a trace s that satisfies $s = \hat{s}\vartheta$, then s can be obtained by the n-th step of $\longrightarrow^t$. Now, we perform a case analysis on the $n+1$-th step.

Case $\langle \hat{s}, a(M).\hat{P}\rangle$: If there exists a step in which $\langle \hat{s}, a(M).\hat{P}\rangle \longrightarrow_p \langle \hat{s}.a(M), \hat{P}\rangle$, and a ground substitution ϑ where $\hat{s}\vartheta$ is a trace, then $M\vartheta$ is a ground message which can be deduced by $s\vartheta$, and its type can be unified by the type of x because of the inference system in Figure C.1. So $\langle s, a(x).P\rangle \longrightarrow \langle s.a(M\vartheta), P\{(M\vartheta)/x\}\rangle$. □

Appendix D

Implemented Maude Codes

D.1 Functions for Refinement Steps

Scanning Parametric Traces

The first function for the refinement step is to analyze a trace. It accepts a parametric trace and a rigid message list as an environment of the trace, and returns the trace's first rigid message, a boolean (true if there exists some rigid message in the trace), and a message list (as an elementary message list ready for unifying the rigid message). The basic strategy is, the function will return the first rigid message and its elementary message list.

```
sort Anares .
  op [_,_,_] : Message Bool Messagelist -> Anares .
  op getMessage : Anares -> Message .
  op getBool    : Anares -> Bool .
  op getMessagelist : Anares -> Messagelist .

  op analyzingTrace : Trace Messagelist -> Anares .
    eq analyzingTrace(Nil, ML1) = [ ERRNAME, false, nil ] .
    ceq analyzingTrace (< LA1 , i , MES1 > . TR1 ,  ML1 ) =
          [getMesRes(getSharedRigid(MES1)), true , ML1 ]
                                if getBoolRes(getSharedRigid(MES1)) .
    ceq analyzingTrace( < LA1, i , MES1 > . TR1 ,  ML1 ) =
                                          analyzingTrace (TR1, ML1)
                             if not getBoolRes (getSharedRigid(MES1)) .
    eq analyzingTrace(< LA1, o , MES1 > . TR1 ,  ML1 )=
                                    analyzingTrace (TR1, MES1 # ML1) .
```

Here we do not show how to define the function `getSharedRigid`, which accepts a message and returns its first rigid sub-message and a boolean (true if the message contains a rigid message).

Due to the differences of rigid message definitions (in Definition 4.3 and Definition 7.4, respectively) and satisfied normal forms in the model for the analysis of authentication and security properties, and for the analysis of non-repudiation and fairness properties, the implementations of this function in two parametric systems are different.

Compared to the function above, not only parametric messages in input actions but also parametric messages in output actions need to be analyzed. The scanning function for non-repudiation and fairness properties is defined as follows.

```
sort Anares .
  op [_,_,_] : Message Bool Messagelist -> Anares .
  op getMessage : Anares -> Message .
  op getBool    : Anares -> Bool .
  op getMessagelist : Anares -> Messagelist .

  op analyzingTrace : Trace Messagelist -> Anares .

   eq analyzingTrace(Nil, ML1) = [ERRNAME, false, nil ] .

   ceq analyzingTrace (< LA1 , i , MES1 > . TR1 ,  ML1 ) =
    [getMesRes(getSharedRigid(MES1, decompose(ML1,ML1))), true , ML1 ]
             if getBoolRes(getSharedRigid (MES1,decompose(ML1,ML1))) .
   ceq analyzingTrace (< LA1, i , MES1 >  . TR1 ,  ML1 ) =
                                           analyzingTrace ( TR1, ML1 )
          if not getBoolRes(getSharedRigid (MES1,decompose(ML1,ML1))) .
   ceq analyzingTrace (< LA1, o , MES1 >  . TR1 ,  ML1 ) =
                                      analyzingTrace ( TR1, MES1 # ML1 )
       if (not getBoolRes(getSharedRigid (MES1,decompose(ML1,ML1)))) or
      (isLocalRigid(getMesRes(getSharedRigid(MES1,decompose(ML1,ML1))),
                                                  getPlayer(LA1) )) .
   ceq analyzingTrace (< LA1, o , MES1 >  . TR1 ,  ML1 ) =
                       [getMesRes(getSharedRigid(MES1,ML1)), true , ML1 ]
             if getBoolRes(getSharedRigid(MES1,decompose(ML1,ML1))) and
               (not isLocalRigid(getMesRes(getSharedRigid
                           (MES1,decompose(ML1,ML1))),getPlayer(LA1))) .
```

Decomposition

According to Definition 4.4, an elementary message set can be obtained by decomposing each pair message and encrypted message whose key is also in the set. We implement a `decompose` function as follows. It accepts two message lists (the latter is used for an environment), and returns a message list. A function `elementary` is defined by applying the `decompose` function.

```
op decompose : Messagelist Messagelist -> Messagelist .
  eq decompose( nil, ML2 ) = nil .
  eq decompose( (MES1,MES2) # ML1, ML2 ) =
              decompose(MES1 # MES2 # ML1, MES1 # MES2 # ML2) .
  ceq decompose( {MES1}k[ name(N1),name(N2) ] # ML1, ML2) =
             {MES1}k[name(N1), name(N2)] # decompose (ML1, ML2)
                           if not in(k[name(N1), name(N2)], ML2) .
  ceq decompose({MES1}k[name(N1), name(N2)] # ML1, ML2) =
               decompose(MES1 # ML1, MES1 # ML2)
```

```
                                  if in(k[name(N1), name(N2)], ML2) .
  eq decompose({MES1}k[MES2, MES3] # ML1,ML2) =
      {MES1}k[MES2, MES3] # decompose(MES1 # ML1, MES1 # ML2) [owise] .
  eq decompose(MES1 # ML1 , ML2) =  MES1 # decompose(ML1, ML2)[owise] .

op elementary : Messagelist -> Messagelist .
  eq elementary (ML1) =  decompose (ML1, ML1) .
```

Unification

A unification function, `unifying`, is used for unifying a rigid message and each message in an elementary message set (defined as a list in the implementation). A unification function accepts two messages and returns a boolean and a substitution. For simplicity, here we only illustrate the unification function by proposing its base cases and some inductive cases. Before defining the unification function, we need an occurrence check function, `oCheck`, which checks whether a parametric variable occurs in a parametric message.

```
sort Result .
  op (_,_) : Substitutions Bool -> Result [ctor] .
  op getSubstitution : Result -> Substitutions .
  op getBool : Result -> Bool .

  op oCheck  : Message Message -> Bool .
    eq oCheck (px(X), px(X)) = true .
    eq oCheck (px(X), (M1,M2)) =
                oCheck(px(X), M1) or oCheck(px(X), M2) .
    eq oCheck (px(X), {M1}M2) =
              oCheck(px(X), M1) or oCheck(px(X), M2) .
    eq oCheck (px(X), k[M1,M2]) =
              oCheck(px(X), M1) or oCheck(px(X), M2) .
    eq oCheck (px(X), Hash[M1]) =
              oCheck(px(X), M1) .
    ... ...
    eq oCheck (M1, M2) = false [owise] .

  op unifying : Message Message -> Result [ comm ] .
    eq unifying(px(X),px(Y)) = (X |-> px(Y), true) .
    eq unifying(px(X),name(Y)) = (X |-> name(Y), true) .
    eq unifying(name(X),name(X))= (nil, true) .
    ceq unifying(px(X),(M1,M2)) = (X |-> (M1,M2), true)
         if not oCheck(px(X), (M1,M2)) .
    ... ...
    eq unifying((M1,M2),(M3,M4)) = ((getSubstitution(unifying(M1,M3)),
      getSubstitution( unifying substitutions
                              (M2,getSubstitution (unifying(M1,M3))),
      substitutions(M4, getSubstitution (unifying(M1,M3))))))),
      ( getBool(unifying(M1,M3)) and getBool(unifying(substitutions
```

```
      (M2, getSubstitution(unifying(M1,M3))),
        substitutions(M4,getSubstitution(unifying(M1,M3))))))) .
  ... ...
  eq unifying(M1,M2) = (nil, false) [owise] .
```

D.2 Protocol Description for the Yahalom Protocol

The protocol description of the Yahalom protocol in the source code mainly has two code fragments. One is to describe behaviors of each principal in the Yahalom protocol, which is represented as follows:

```
crl [A_1] : < [ TR1 ], SUBLIST, ot >  =>
 < [ (TR1 . < a(1), o, (name(0), name(10) ) >) ], SUBLIST, ot >
                      if not labelinTrace (TR1, a(1)) .
crl [A_2] : < [ TR1 ], SUBLIST, ot > =>
   < [ TR1 .  < a(2), i, ((px(0), {(name(1), px(1)), name(10)}
    k[name(1),name(0)]), px(2)) > .
         < a(3),o,(px(2), {px(0)}px(1)) > ], SUBLIST, ot >
   if labelinTrace(TR1, a(1)) and  not labelinTrace(TR1,a(2)) .
crl [A'_1] : < [ TR1 ],SUBLIST,ot > =>
 < [ (TR1 . < a'(1), o, (name(30), name(39) ) >) ], SUBLIST, ot >
   if not labelinTrace(TR1,a'(1)) .
crl [A'_2] :< [ TR1 ],SUBLIST, ot > =>
    < [ TR1 .  < a'(2), i, ((px(30), {(name(31), px(31)), name(39)}
    k[name(31),name(30)]),px(32)) > .
    < a'(3), o, (px(32), {px(30)}px(31)) > ],SUBLIST,ot >
    if  labelinTrace(TR1, a'(1)) and not labelinTrace(TR1,a'(2)) .
crl [B_1] :< [ TR1 ],SUBLIST,ot > =>
    < [ (TR1 . < b(1), i, (name(0), px(10)) > .
    < b(2), o, ((name(1),name(11)),
   {name(0),px(10)}k[name(1),name(2)]) > )], SUBLIST, ot >
                if not labelinTrace (TR1,b(1)) .
crl [B_3] : < [ TR1 ],SUBLIST,ot > =>
    < [ (TR1 . < b(3), i, ({(name(0), px(11)), name(11)}
    k[name(1),name(2)], {name(11)}px(11) ) > .
    < acc, o, ({(name(0), px(11)), name(11)}k[name(1),name(2)],
                            {name(11)}px(11) ) >) ],SUBLIST,ot >
   if labelinTrace(TR1, b(1)) and labelinTrace(TR1, b(2)) and
                                  not labelinTrace (TR1, b(3)) .
crl [B'_1]: < [ TR1 ],SUBLIST,ot > =>
     < [ (TR1 . < b'(1), i,(px(27), px(20)) > .
     < b'(2), o, ((name(1),name(21)),{px(27), px(20)}
         k[name(1),name(2)]) > )], SUBLIST, ot >
                              if not labelinTrace (TR1, b'(1)) .
crl [B'_3] : < [ TR1 ],SUBLIST,ot > =>
     < [ (TR1 . < b'(3), i,({(px(27), px(21)), name(21)}
     k[name(1),name(2)],{name(21)}px(21)) > ) ], SUBLIST, ot >
```

```
    if labelinTrace (TR1, b'(1)) and  labelinTrace (TR1, b'(2)) and
                                    not labelinTrace (TR1, b'(3)) .
  crl [S_1] : < [ TR1 ],SUBLIST,ot > =>
     < [ (TR1 . < s(1),i,((px(20),px(21)),
                     {px(22),px(23)}k[px(20),name(2)]) > .
     < s(2), o,((px(21),{(px(20), k[name(0),name(1)]),px(23)}
      k[px(22),name(2)]),{(px(22), k[name(0),name(1)]),px(21)}
                           k[px(20),name(2)]) >) ],SUBLIST, ot >
                                  if not labelinTrace (TR1, s(1)) .
```

The other fragment is to describe the authentication specification for the Yahalom protocol, which is represented as follows:

```
search [1] in YAHALOMPROTOCOL :init=>*

   < [ TR1 ], NIL, st > such that not ( labelinTrace(TR1, acc)
     implies(
       ( (labelinTrace(TR1, a(3)) and labelbefore (TR1, a(3),acc))
        and equal((getLabelMessage(TR1,acc)),(getLabelMessage(TR1,a(3))))
       )
     )
    ) .
```

D.3 Protocol Description for the RA protocol

The protocol description of the recursive authentication protocol (in Subsection 3.4.2) has also two code fragments. One is to describe behaviors of each principal in the protocol, which is represented as follows:

```
    crl [A_1] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
                  < ML1, [ (TR1 . < a(1), o, H( lk[MA,name(1)],
                  (((MA,A[MA]),MN),null)) >) ], STACK, SUBLIST, oc >
                            if noc labelinTrace (TR1, a(1)) .
    crl [A_2] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
                  < ML1, [ TR1 .
                  < a(2), i, {((px(11),A[MA]),MN)}lk[MA,name(1)] > .
                   < acc, o, {((px(11),A[MA]),MN)}lk[MA,name(1)] >
                                        ], STACK, SUBLIST, oc >
                 if  labelinTrace (TR1, a(1)) and
                                        noc labelinTrace (TR1, a(2)) .

    crl [B_1] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
                  < ML1, [ ( TR1 .
                    < b(1), i, H(lk[MA,name(1)],
                                 (((MA,A[MA]),px(21)),px(22)) )  > .
                    < b(2), o, H(lk[A[MA],name(1)],
                      (((A[MA],A[A[MA]]),N[px(21)]),H(lk[MA,name(1)],
```

```
                                (((MA,A[MA]),px(21)),px(22)) )) ) >
                          ) ], push (STACK),  SUBLIST, oc >
                                if noc labelinTrace (TR1, b(1)) .
    crl [B_3] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
                < ML1, [ ( TR1 . < b(1), i, H(lk[MA,name(1)],
                                  (((MA,A[MA]),px(21)),px(22)) )  > .
                   < b(3), o, H(lk[A[MA],name(1)],
                     (((A[MA],name(1)),N[px(21)]),H(lk[MA,name(1)],
                             (((MA,A[MA]),px(21)),px(22)))) ) >
                             ) ], pop (STACK) ,  SUBLIST, oc >
                                if noc labelinTrace (TR1, b(1)) .

    crl [B_5] : < ML1, [ TR1 ], STACK, SUBLIST, oc >  =>
            < ML1, [ ( TR1 .
              < b(4), i,(({{(px(24),px(25)),N[px(21)]}}lk[A[MA],name(1)],
                 {{(px(25),MA),N[px(21)]}}lk[A[MA],name(1)]), px(26)) > .
                < b(5), o, px(25) > ) ],  STACK, SUBLIST, oc >
                                if labelinTrace(TR1, b(1)) and
                                          noc labelinTrace (TR1, b(4)) .


    crl [S_1] : < ML1, [ TR1 ], STACK,  SUBLIST, oc >  =>
            < (({{(k[Mk],name(1)),px(32))}}lk[px(31),name(1)],
                           {{(Mk,px(33)),px(32))}}lk[px(31),name(1)]),
            {{(Mk,px(31)),px(34))}}lk[px(33),name(1)] ) # ML1,
                           [ (TR1 . < s(1), i, H (lk[px(31),name(1)],
                      (((px(31),name(1)),px(32)), H(lk[px(33),name(1)],
                       (((px(33),px(31)),px(34)),null) )))  > .
             < s(2), o, (({{(k[Mk],name(1)),px(32))}}lk[px(31),name(1)],
                       {{(Mk,px(33)),px(32))}}lk[px(31),name(1)]),
                          {{(Mk,px(31)),px(34))}}lk[px(33),name(1)] ) >
                            ) ], STACK,  SUBLIST, oc >
                                if noc labelinTrace (TR1, s(1)) .
```

The other fragment is to describe the authentication specification (see Characterization 6.1) for the recursive protocol, which is represented as follows:

```
search [1] in RECURSIVEPDS :
  init =>* < ML1, [ TR1 ], STACK, NIL, ec > such that not (
    labelinTrace(TR1, acc) implies (
        ( labelbefore (TR1, b(2), acc)) and
            equal ((getLabelMessage (TR1,acc)),
                                  (getLabelMessage (TR1, b(2))))
     )
  )     .
```

D.4 Protocol Description of FAIRO for the Simplified ZG Protocol

The protocol description of the FAIRO for the simplified ZG protocol in the source code has also two code fragments. One is to describe behaviors of each principal in simplified ZG protocol, which is represented as follows:

```
crl [A_1] : < [ TR1 ], SUBLIST, ot >  =>
          < [ (TR1 . < label(name(0),1), o, {((name(1),name(11)),
                          px(12))}-k[name(0)] >) ], SUBLIST, ot >
                       if not labelinTrace (TR1, label(name(0),1)) .
crl [A_2] : < [ TR1 ], SUBLIST, ot >  =>
        < [ (  TR1 .  < label(name(0),2), i, {((name(0),name(11)),
       px(12))}-k[name(1)] > .  STOP(name(0)) ) ], SUBLIST, ot >
                    if  labelinTrace (TR1, label(name(0),1)) and
                        not labelinTrace (TR1, label(name(0),2)) .
crl [A'_2] : < [ TR1 ], SUBLIST, ot >  =>
           < [ TR1 .  < label(name(0),2), i, {((name(0),name(11)),
             px(12))}-k[name(1)] > . < label(name(0),3), o,
          {((px(17),px(18)),px(19))}-k[name(0)] > ], SUBLIST, ot >
                 if  labelinTrace (TR1, label(name(0),1)) and
                        not labelinTrace (TR1, label(name(0),2)) .
crl [A_3] : < [ TR1 ], SUBLIST, ot >  =>
     < [ ( TR1 . < label(name(0),4), i, {(((name(0),name(1)),
     name(11)),px(19) )}-k[name(2)] > .  < evidb, o, ( {((name(0),
     name(11)),px(12))}-k[name(1)], {(((name(0),name(1)),name(11)),
                       px(19) )}-k[name(2)] ) > ) ], SUBLIST, ot >
                if not labelinTrace (TR1, label (name(0),4)) and
                             not stopinTrace (TR1, name(0)) and
                            labelinTrace(TR1, label(name(0),3)) .

crl [B_1] : < [ TR1 ], SUBLIST, ot >  =>
       < [ ( TR1 . < label(name(1),1), i, {((name(1),px(23)),
          px(24))}-k[name(0)] > . STOP(name(1)) ) ] , SUBLIST, ot >
                     if not labelinTrace (TR1, label (name(1),1)) .
crl [B'_1] : < [ TR1 ], SUBLIST, ot >  =>
 < [ ( TR1 . < label(name(1),1), i, {((name(1),px(23)),px(24))}
 -k[name(0)] > . < label(name(1),2), o, {((name(0),px(25)),px(26))}
                                  -k[name(1)] > ) ] , SUBLIST, ot >
                     if not labelinTrace (TR1, label (name(1),1)) .
crl [B_2] : < [ TR1 ], SUBLIST, ot >  =>
 < [ ( TR1 . < label(name(1), 3), i,  {(((name(0),name(1)),px(23)),
px(27) )}-k[name(2)] > . <  evida, o, (  {((name(1),px(23)),px(24))}
-k[name(0)],{(((name(0),name(1)),px(23)),px(27) )}-k[name(2)] ) >) ],
                                                    SUBLIST, ot >
               if labelinTrace (TR1, label(name(1),2)) and
              not labelinTrace (TR1, label(name(1),3)) and
```

```
                                        not stopinTrace(TR1,name(1)) .

    crl [S_1] : < [ TR1 ], SUBLIST, ot >  =>
      < [ ( TR1 . < label(name(2),1), i, {((px(31),px(32)),px(33))}
      -k[px(34)] > . < label(name(2),2), o, {(((px(34),px(31)),px(32)),
     px(33))}-k[name(2)] > . < label(name(2),3), o, {(((px(34),px(31)),
                     px(32)),px(33))}-k[name(2)] > ) ],  SUBLIST, ot >
                     if not labelinTrace (TR1, label(name(2),1)) .
```

The other fragment is to describe the FAIRO specification (see in Characterization 7.3) for the simplified ZG protocol, which is represented as follows:

```
search [1] in ZGPROTOCOL : init =>* < [ TR1 ], NIL, et > such that
not (
    labelinTrace(TR1, evida) and
      getBool (unifying ( getLabelMessage(TR1,evida),
           ({((name(1),px(71)),px(72))}-k[name(0)],{(((name(0),name(1)),
                                      px(71)),px(73))}-k[name(2)])) )
     implies(
         labelinTrace(TR1, evidb) and
        equal (
          getLabelMessage(substitutingTrace(TR1, getSubstitution
                                (unifying ( getLabelMessage (TR1,evida),
           ({((name(1),px(71)),px(72))}-k[name(0)],{(((name(0),name(1)),
                    px(71)),px(73))}-k[name(2)]))))), evidb) ,
          substitutions(({((name(0),px(71)),px(72))}-k[name(1)],
                     {(((name(0),name(1)),px(71)),px(73))}-k[name(2)]),
             getSubstitution(unifying ( getLabelMessage (TR1,evida),
           ({((name(1),px(71)),px(72))}-k[name(0)],{(((name(0),name(1)),
                                      px(71)),px(73))}-k[name(2)]))))
       )
       )
     ).
```