IMPROVED SPI-CALCULUS FOR REASONING ON CRYPTOGRAPHIC PROTOCOLS

"Master Thesis"

By

Saleh Mansour Bani Hani.

Supervisors

Dr. Hasan Al-Refai. Dr. Mourad Maouche.

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia university

June 2012

لغة ال Spi الرياضية محسنة لتحليل بروتوكولات التشفير

إعداد
صالح منصور بني هاني

المشرف
د. حسن الرفاعي

المشرف المشارك
د. مراد معوش

قدمت هذه الرسالة استكمالا لمتطلبات الحصول على درجة الماجستير في علم الحاسوب.

عمادة البحث العلمي والدراسات العليا

جامعة فيلادلفيا

حزيران ، 2012

IMPROVED SPI CALCULUS FOR REASONING ON CRYPTOGRAPHIC PROTOCOLS

By

Saleh Mansour Bani Hani

Supervisor
Dr. Hasan Al-Refai

Co-Supervisor
Dr. Mourad Maouche

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree In Computer Science.

Deanship of academic Research and Graduate Studies

Philadelphia University

June , 2012

DEDECATION

I dedicate this work to

my beloved family

AKNOWLEDGMENT

I would like to thank all those people whose were true sources of inspiration, knowledge, guidance and help to me throughout the period of my master research. In particular, my extreme appreciation and thanks goes to my supervisors Dr. Hasan Al-Refai and Dr. Morad Mouach for their constant encouragement, guidance, motivation and support. My deepest appreciation goes to all my teachers at the department of Compute Science, and special thanks to Philadelphia University for giving me the chance of continue my studies.

Also, my final words are reserved to my family: my father, my mother, my sisters and my brother. Thank you all for the love and support.

Saleh Bani-Hani

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

- Latin Symbols -

Α	α	alpha
Β	β	beta
Γ	γ	gamma
Δ	δ	delta
Ε	ε	epsilon
Ζ	ζ	zeta
Η	η	eta
Θ	θ	theta
Ι	ι	iota
Κ	κ	kappa
Λ	λ	lambda
Μ	μ	mu
Ν	ν	nu
Ξ	ξ	xi
Ο	o	omnicron
Π	π	pi
Ρ	ρ	rho
Σ	σ	sigma
Τ	τ	tau
Υ	υ	upsilon
Φ	ϕ	phi
Χ	χ	Chi
Ψ	ψ	Psi
Ω	ω	Omega

لغة ال Spi الرياضية محسنة لتحليل بروتوكولات التشفير

الملخص

تتعرض معظم بروتوكولات التشفير لاختراقات متنوعة فعمل الكثير من الباحثين على تطوير أدوات تصميم و تحليل مثل هذه البروتوكولات, و ذلك من أجل التحقق من ضمان الخواص الأمنية مثل التوثيق و السرية. و من أكثر الأدوات المثنية بفعاليتها هي المنهجيات الشكلية و ذلك لدقتها و صلابتها في تصميم و تحليل مثل هذه البروتوكولات.

تعتبر لغة ال spi الرياضية المقدمة من الباحثين العبادي و غوردون كاستكمال للغة ال Pi الرياضية باضافة عمليات التشفير,لاستخدامها تصميم و تحليل بروتوكولات التشفير.

يوجد الكثير من الباحثين الذين عملوا على تحسين هذه اللغة, حيث قاموا بوضع الكثير من الفرضيات المعرفة مسبقا لتقييد بيئة العمليات و دون التطرق للتعامل مع النظم الحقيقية التي لا تتطابق مع تلك الافتراضات و المقيدات.

في هذه الأطروحة، قدمنا لغة ال Ph-spi الرياضية كلغة محسنة للغة ال spi الرياضية و ذلك لتكون خطوة نحو لغة رمزية و الآلية. لغة ال Ph-spi الرياضية حيث تمتاز بقدرتها على تقييم كل حدث في العمليات القائمة و تمكن من اتخاذ القرار المناسب للتعامل مع البيئات مفتوحة.

حيث قمنا باستتحداث دالة تقييم قادرة على على تحليل البروتوكولات مطبقة فعليا في بيئة مفتوحة, مع القدرة على اتخاذ القرارات المناسبة للتعامل مع بيئة مفتوحة متغيرة السلوك, كذلك قادرة على تحقق من الإثبات الخصائص الأمنية الرئيسية مثل التوثيق والسرية.

كما أن, بناء الرسائل في لغة ال ,Ph-spi الرياضية تكون مهيكلة للتمكن من احتواء مجموعة من المكونات في رسالة رمزية واحدة. كما أنها تشمل على كل من الطابع الزمني، ودالة البعثرة، التوقيع الرقمي والتشفير غير المتناظر, كما هو الحال في جميع البروتوكولات التطبيقية الحقيقية.

IMPROVED SPI CALCULUS FOR REASONING ON CRYPTOGRAPHIC PROTOCOLS

Abstract

Most of cryptographic protocols are subjects to very subtle attacks. Therefore, many researchers have developed tools to model and analyze protocols to guarantee their security properties.

Among these developed tools, the spi calculus has proved to be a powerful formal methods have useful for analyzing and reasoning on cryptographic protocols.

However, such research work assumed that the environment will be restricted to some predefined rules and assumptions without solving the case of interacting with real systems that matched with such restrictions.

In this thesis, we introduced an improved version of spi calculus called the Ph-spi calculus. Such calculus provides the ability for evaluating each action in the running processes and making suitable decision to be more suitable with open environment.

Therefore, we have enhanced the evaluation function to make it capable for making suitable decisions to handle the open and changeability in environment behaviors as well as evaluating and validating every action in protocol processes for proving the main security properties as authentication and confidentiality.

In the Ph-spi calculus, the message is structured to be the same as in real protocols that have a tuple of messages. Also, it includes all operators needed for such protocols such as timestamp, hash function, digital signature and asymmetric key cryptosystem. Hence, Ph-spi calculus ready is to be used for real protocols such as e-commerce protocols.

Keywords: Cryptographic protocol, Cryptographic protocol analysis, spi calculus, evaluation function, testing equivalence.

Chapter I:

INTRODUCTION

1.1 *Background*

The Handbook of Applied Cryptography defines communication protocol as a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective (A. Alfred, et al. 2001).

Over the years, the problems of network security have become more and more complex. Solving these problems usually requires the design of protocols that rely on cryptographic systems to hide some secret information: these protocols are "cryptographic."

The use of cryptographic protocols is widely spread – Transport Layer Security (TLS, formerly SSL) and Secure Shell (SSH) are common cryptographic protocol used to secure connections over the Internet. However, they assumed to be secured by protocols using cryptographic methods as encryption or digital signatures, promising security properties as secrecy or authentication.

Establishing security in these applications can be divided into two main areas, cryptography which is the art and science of writing a text encrypted by a key to prevent an attacker from the ability to access and modify the plaintext. The second area is the *cryptographic protocol,* also known as *security protocols,* that is used to achieve the authentication of agents or nodes, establishing session keys between nodes, ensuring secrecy, integrity, non-repudiation, and authentication properties. These protocols involved on the exchange of messages between nodes, often requiring the participation of a trusted third party or session server. Typically they make liberal use of various cryptosystems, such as symmetric and asymmetric encryption, hash functions, and digital signatures and timestamps are also used.

1.2 *Cryptographic Protocols*

Cryptographic protocols are protocols that use cryptography to distribute keys and authenticate principals and data over a network. The network is usually assumed to be hostile, in that it may contain intruders who can read, modify, and delete traffic, and who may have control of one or more network principals. A cryptographic protocol must be able to achieve its goals in face of these hostile intruders. Because of this, such protocols are often subject to nonintuitive attacks which are not easily apparent even to a careful inspector. Many of these attacks do not depend upon any flaws or weaknesses in the underlying cryptographic algorithm, and can be exploited by an attacker who is able to do no more than the basic operations listed above, but in any arbitrary order. Other attacks may depend upon subtle properties of the cryptographic algorithms, or on statistical analysis of message traffic, or on some combination of the above.

Designing and analyzing cryptographic protocols is a very challenging problem. In open environment, such as the Internet, protocols should work even under worst-case assumptions, namely messages may be eavesdropped or tampered with by an attacker (also called the intruder or adversary) or dishonest or careless principals. Surprisingly, severe attacks can be conducted even without attacking and breaking cryptography, but rather by attacking communication itself. These attacks exploit weaknesses in the protocol's design whereby protocols can be defeated by cleverly manipulating and replaying messages in ways not anticipated by the designer. (S. Matsuo et al. 2009) This includes attacks such as: man-in-the-middle attacks, where an attacker is involved in two parallel executing sessions and passes messages between them; replay attacks, where messages recorded from previous sessions are played in subsequent ones; reflection attacks, where transmitted information is sent back to the originator; and type flaw (confusion) attacks, where messages of different types are substituted into a protocol (e.g., replacing a name with a key). Typically, these attacks are simply overlooked, as it is difficult for humans, even by a careful inspection of simple protocols, to determine all the complex ways that different protocol sessions could be interleaved together, with possible interferences coming from a malicious intruder (S. Matsuo et al. 2009).

Historically, the security analysis of cryptographic protocols has consisted of testing their resistance to different attacks imagined by the designer. A protocol could be believed very secure at the start and, after that, modified several times because of the discovery of some weaknesses. This type of analysis does not prove the protocol security. Recently, some methods for formally verifying the security of cryptographic protocols have begun to be researched but, in most cases, they can only be applied to special kinds of protocols (I. Fléchais. 2005).

What is needed are methods to speed up the development and analysis of cryptographic protocols. Moreover, if these methods are to be used to certify protocols, then they must be mathematically precise, so that exact statements are possible about the scope and significance of the analysis results. This role can be filled by formal methods.

Over the last two decades, the security community has made substantial advances in developing formal methods for analyzing cryptographic protocols and thereby preventing the kinds of attacks mentioned above.

In recent years, formal methods have been proven to be a useful in the area of analysis cryptographic protocols because its allow one to do overall analysis which covers different paths the attackers can take.

1.3 *Formal Methods*

A formal method owes much to the security community among other fields of appliances. In the United States, the National Security Agency was a major source of funding in the 70s and early 80s for formal methods research and development. Results included the development of formal security models, tools for reasoning about security, and applications of these tools to proving systems secure. In return, security provided a challenging research application for the formal methods utilization.

Until the late 1980s, cryptographic protocols were only analyzed by informal reasoning. If a protocol claimed to achieve the goal of confidentiality, researches were prompted to study the protocol in detail and decided whether this is true (M. Tschantz and J. Wing. 2009).

The literature shows that formal approaches can significantly help to detect protocol flaws, as well as to yield general principles of secure protocol design. Some approaches lack expressiveness or automation; others are just too complicated to use on realistic protocols. Informal reasoning indeed retains its importance: it is crucial to grasp the semantics of a protocol design beyond its bare representation as a sequence of messages; it may find simple flaws and minor weaknesses of a protocol more quickly than formal reasoning; it is easier to follow for a non-experienced audience; it helps to develop formal approaches. For more details concerning the process algebra and oprtational semantics you can refer to Appendix A.

1.4 *Process Environment*

Many research works have been attempted to develop an operational semantics to evaluate the interactions between protocol processes and the environment. Some of them had built their work to fit with closed environment with special case protocols (R. Focardi and M. Maffei, 2004, Y. Gu et al, 2005, A. Tiu and J. Dawson, 2010) others construct their work with a predefined environment (restricted environment) (H. Al-Refai 2009, J. Borgstrom, 2009). Unfortunately, no work assumed the open environment.

This section provides a detail definition for each type of these environments and what protocols are suitable to use.

1.4.1 *Closed Environment*

The processes cannot trigger any action out of their scope or boundary. This means that the channels are initiated to be in that boundary and no one outsider can interact with these channels. *Figure*-1.4.1.1 shows such environment.

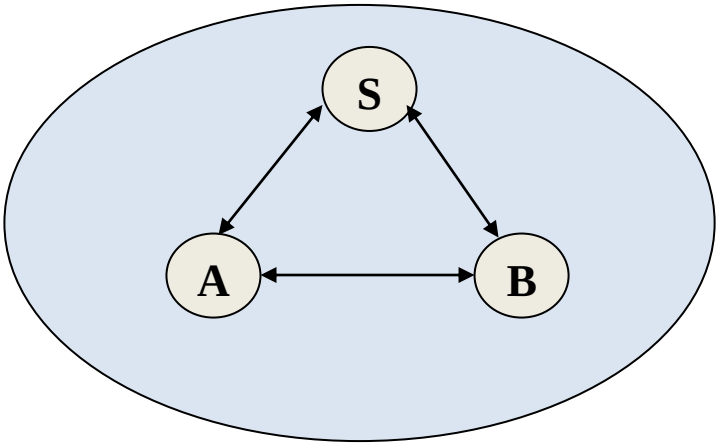

Figure 1.4.1.1 Closed Environment

Here, the sender A, the server S and the receiver B used restricted channels ch_{AS}. ch_{AB} and ch_{SB} between them. These channels cannot be accessed by outsiders.

1.4.2 *Restricted Environment*

It differs from the closed environment by that the outsiders can get access to the channels but with predefined rules and assumptions. Means that there is a possible channels used to allow the processes in the protocol to interact with the environment out of the scope or boundary. *Figure*-1.4.2.1 shows such case.

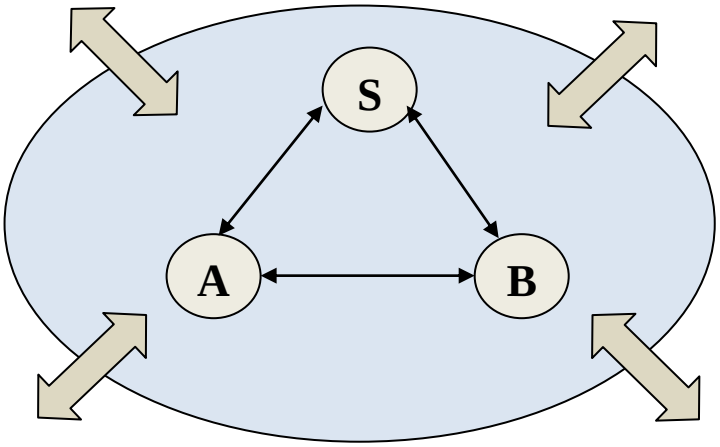

Figure-1.4.2.1 Restricted Environment

1.4.3 *Open Environment*

From the environment types mentioned above, it seem clear that the security controls can achieved easily, since the environment are already defined and the interactions with the outsider is not allowed or restricted.

In the case of an open environment the channels are public as in real protocols used in the Internet. Means they are interacts with all processes in the environment without closing or restricting them. Such type of environment needs for additional functions to be built in the syntax and the operational semantics to trace each action taken by the processes then to decide wither to proceeds or terminate that process in case of any attack. *Figure*-1.4.3.1 show that the interactions with the environment is open and channels are public for use of all.

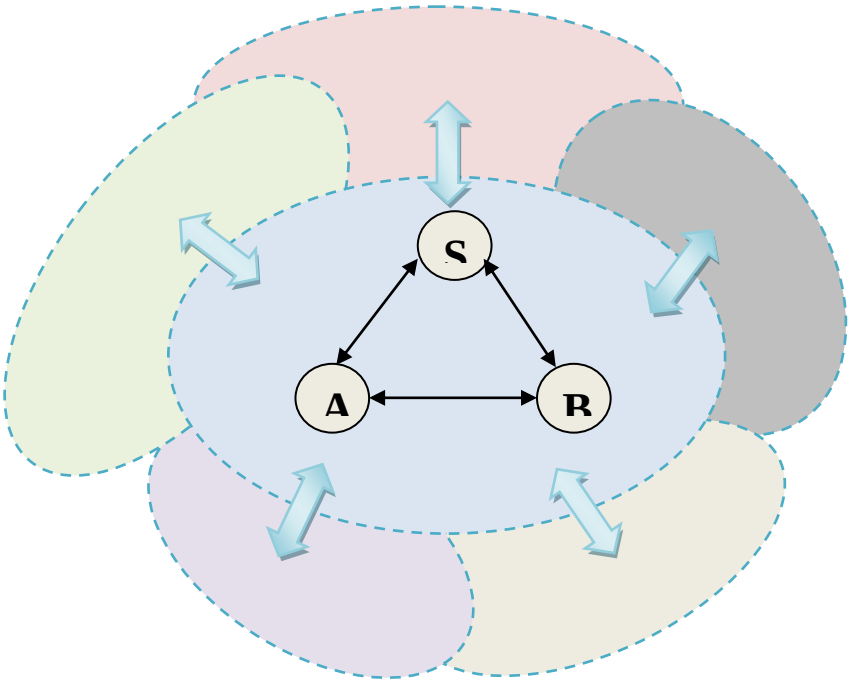

Figure 1.4.3.1 Open Environment

The behaviors of such type of environment are changeable where the knowledge can be observed from the process actions in the protocol.

These protocols often rely on special security tokens such as numbers used only once, called nonces, or fresh keys generated by principals of the protocol the environment assumed to unknown them. In closed and restricted environment, this case is approved since all actions are controlled in that environment.

Moreover, as the size of the messages exchanged in such a protocol and thus the recursion depth of the principals' computations is not explicitly bounded by the protocol's description that needs a mechanism to provide the principals with the possibility to generate an unbounded number of nonces and fresh keys.

In coming section (1.5), we will give a simple example to show how protocol processes can be described by π-calculus. We use π-calculus as a basis to spi calculus.

1.5 *Simple Example using π-calculus*

The π-calculus is a model of computation for concurrent systems (R. Milner et al, 1992). The syntax of π-calculus lets represent processes, parallel composition of processes, synchronous communication between processes through channels, creation of fresh channels, replication of processes, and nondeterminism.

We will give the definitions of these processes, then a simple example will be used to show their roles.

A *process* is an abstraction of an independent thread of control. A *channel* is an abstraction of the communication link between two processes. Processes interact with each other by sending and receiving messages over channels.

Let P and Q denote processes. Then

- $P \mid Q$ denotes a process composed of *P* and *Q* running in parallel.
- $a(x).P$ denotes a process that waits to read a value *x* from the channel *a* and then, having received it, behaves like *P*.
- $\bar{a}(x).P$ denotes a process that first waits to send the value *x* along the channel *a* and then, after *x* has been accepted by some input process, behaves like *P*.
- $(va).P$ ensures that *a* is a fresh channel in *P*. (Read the Greek letter "nu" as "new.")
- $!P$ denotes an infinite number of copies of *P*, all running in parallel.
- $P + Q$ denotes a process that behaves like either *P* or *Q*.
- **0** denotes the inert process that does nothing.

All concurrent behavior that you can imagine would have to be written in terms of just the above constructs.

The coming example had been used by (M. Abadi and A. Gordon 1997, 1998). Channels can be established in different ways. In this example we describe the Wide Mouthed Frog protocol as the abstract level, which has the basic structure shown in Figure 1.5.1.

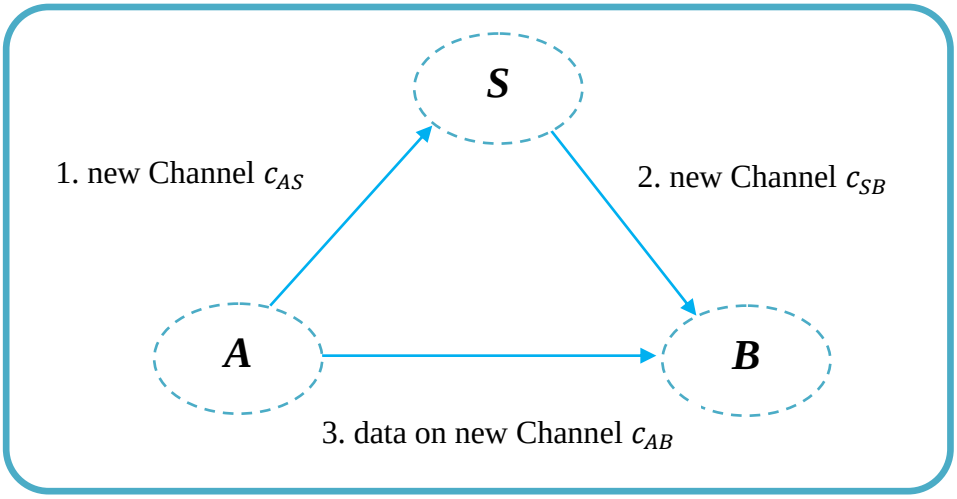

Figure 1.5.1: The Wide Mouthed Frog protocol

This version is abstract in that we deal with channels instead of keys; it is simplified in that channel establishment and data communication happen only once. In chapter III we show how to treat keys and how to allow many instances of the protocol, with an arbitrary number of messages.

Informally, we may write this protocol as follows:

Message 1 $A \rightarrow S : c_{AB}$ on c_{AS}

Message 2 $S \rightarrow B : c_{AB}$ on c_{SB}

Message 3 $A \rightarrow B : M$ on c_{AB}

Here c_{AS} is a channel that *A* and *S* share initially, c_{SB} is a channel that *S* and *B* share initially, and c_{AB} is a channel that *A* creates for communication with *B*. After passing the channel c_{AB} to *B* through *S*, *A* sends a message *M* on c_{AB}. Note that *S* does not use the channel, but only transmits it.

Using π-calculus, we formulate this protocol as follows:

$$A(M) \;::= (\nu c_{AB})\overline{c_{AS}}(c_{AB}) \; on \; (c_{AS})$$

$$S \;::= c_{AS}(x).\overline{c_{SB}}(x)$$

$$B \;::= c_{SB}(x).x(y).F(y)$$

$$Inst(M) \;::= (\nu c_{AS})(\nu c_{SB})(A(M) \mid S \mid B)$$

Here, we write *F*(*y*) to represent what *B* does with the message y that it receives. The restrictions on the channels c_{AS}, c_{SB}, and c_{AB} reflect the expected privacy guarantees for these channels. The most salient new feature of this specification is the use of scope extrusion: *A* generates a fresh channel c_{AB}, and then sends it out of scope to *B* via *S*.

For discussing authenticity, we introduce the following specification:

$$A(M) \;::= (\nu c_{AB})\overline{c_{AS}}(c_{AB}).\overline{c_{AB}}\langle M\rangle$$

$$S \;::= c_{AS}(x).\overline{c_{SB}}(x)$$

$$\underset{Spec}{B} \;::= c_{SB}(x).x(y).F(M)$$

$$\underset{Spec}{Inst}(M) \;::= (\nu c_{AS})(\nu c_{SB})(A(M) \mid S \mid \underset{Spec}{B}(M))$$

According to this specification, the message *M* is communicated “magically": the process *F* is applied to the message *M* that *A* sends independently of whatever happens during the rest of the protocol run.

Proposition 1.5.1: We obtain the following authenticity and secrecy properties (M. Abadi and A. Gordon, 1998):

Authenticity: $Inst(M) \cong \underset{Spec}{Inst}(M),\ for\ all\ M.$

Secrecy: $Inst(M) \cong Inst(M')\ if\ \ F(M) \cong F(M'),\ for\ all\ M\ and\ M'.$

Again, these properties hold because of the scoping rules of the π-calculus.

We believe that the example just given is rather encouraging. It indicates that the π-calculus is a natural language for describing some security protocols. In particular, the restriction operator and scope extrusion allow convenient representations for the possession and communication of channels.

We do not wish to claim that the π-calculus did not enable us to describe all security protocols, even at an abstract level. For example, some protocols rely on asymmetric channels (channels of the kind implemented with public-key cryptography). It may be possible to represent such asymmetric channels in the π-calculus but extending the π-calculus may be simpler and more effective. However, the restriction operator and scope extrusion should be useful for describing security protocols even in extensions of the π-calculus.

Nevertheless, the π-calculus is yet too abstract to explicitly represent cryptographic operations that are an important ingredient of security protocols. To fill this gap, the spi calculus (M. Abadi and A. Gordon, 1998) defined as an extension of the π-calculus with primitives for encryption and decryption.

The *spi calculus* introduced by M. Abadi and A. Gordon as an extension of the π-calculus with cryptographic primitives has been used for cryptographic protocols modeling. In the spi calculus the protocols are stated as processes and the properties are proven using notions of protocols equivalence, (Abadi and Gordon 1998).

Many researchers tried to enhance the original work of Abadi and Gordon (M. Abadi and A. Gordon 1998) such as (R. Focardi and M. Maffei, 2004, Y. Gu et al, 2005, A. Tiu and J. Dawson, 2010) but their works cannot be applied on real protocols but on research cases protocols, where the most researchers for spi calculus focused on these types of protocols.

In this thesis, we have introduced a improved version of spi calculus; which we called Ph-spi calculus, with an enhanced syntax and semantics that is capable of representing real protocol running in an open environment.

The analysis of protocols in such type of environment cannot be built according to pre-defined assumptions and restrictions that enforce the interactions between the processes presenting the protocol and processes presenting such environment to be bounded to these definitions.

However, most of the research works in the analysis of cryptographic protocols (R. Focardi and M. Maffei, 2004, Y. Gu et al, 2005, A. Tiu and J. Dawson, 2010); including the research work in spi calculus, assumed the environment as an arbitrary process to run in restricted mode, while the real environment is inherently open by

nature; providing equal availability of knowledge; without restriction of freedom of thought or information. Also, such environment is more complex to be managed or controlled by set of restriction and assumption.

The closed and restricted environments consider the correspondence between two different process pairs for their equivalence to prove the confidentiality and integrity properties without considering the unexpected behavior in the real case of environment including the number of transitions and the expanding in running processes. From the work in the literature, what they have to do is to restrict that environment to behave as their model needs without solving the case of interacting with real systems that matched with such restricted assumptions. Some of these restrictions concern the used channels where the channel cannot be restricted to only specific participants or couldn't be assumed as a same for all of them. Also, during the expansion of the processes, the processes concern generation of a fresh names for replacing all free occurrences in internal actions taken in transition, these names assumed to be generated for once, in case of extended process the names should be freshly generated with each extend and evaluated everywhere they can be used.

Some of resent works had been attempt to conjugate the role of the environment with the running processes to integrate a protocol as a system. Hence, a process pair of *(P, Q)* is drawn to be further analyzed in terms of the triples *(e, P, Q)* of process pair and an environment argument.

Therefore, when two processes can resist the same attacks from the environment, they argue that they are testing equivalent and should in some way be indistinguishable to the attackers that is to prove the confidentiality and integrity properties. But still the environment defined according to their set of assumptions.

So, the improved Ph-spi calculus use to construct an automated function in order to evaluate each action in the running processes for deciding a suitable reaction for any

emerged changes in the knowledge of the environment, this evaluation function help us to get free from these restrictions that exist in previous works and give the Ph-spi calculus the ability of running in an environment where the behavior of such environment is unpredictable.

1.6 *Research Motivation*

The current spi calculus did not proved to be suitable for real protocols or even capable to interact with an open environment.

Therefore, our motivation is to introduce an improved version of spi calculus called Ph-spi calculus capable to reasoning on the real protocols running in an open environment. For that, we have constructed an evaluation function that used for evaluating and validating every action in protocol processes and making suitable decisions during the interaction with open environment. Also it is a powerful descriptive calculus for proving the main security properties as authentication and confidentiality .

The improved calculus ready to be used for real protocols such as e-commerce protocols, which it includes all operators needed for such protocols such as timestamp, hash function, digital signature and asymmetric key cryptosystem as well as the messages are structured to be same as in real protocols that have a tuple of messages. It is used to trace and analysis such protocols for deciding there correctness.

1.7 *The Problem Statement*

Current spi calculus suffered from several limitations, these limitations due to number of restrictions and assumptions they have built in their work in order to reach specific

approaches for a specific type of environment and protocols. Theses limitations are briefly discussed by the following:

I. The current spi calculus dealt with transferring a single unstructured message for sending each message in a single action, where a tuple of messages is mostly needed in implementing real protocols. Such as, for sending timestamp, user identity, user certificate and its validation date from the user to the merchant all in one output action.

II. The encryption/decryption operations assumed to not disclose any of their knowledge to the environment. This assumption in case of closed and restricted environment.

III. The current researchers (H. Al-Refai, 2009, J. Borgstrom, 2009) did not provide suitable solutions for input transitions, not even for finite processes. The tracing and evaluation for each of those inputs is not considered.

IV. The freshness of generated names and variables (such as channels, variables used in mapping for substitution, keys, timestamp …etc.) assumed to be bonded in a running process. This in case of dealing with known environment (closed or restricted) and they restrict these names and variables to be bounded without any evaluation to validate if the environment observe or learn any knowledge for those names and variable. This case will be discussed in chapter III Definitions (3.2.1.1), (3.2.1.4), (3.2.1.5), (3.2.1.6) and (3.2.1.7).

V. Some researchers added a Boolean guard, but they did not evaluate the formula to guarantee its correctness (H. Al-Refai, 2009, J. Borgstrom, 2009).

The limitations shown above define the following set of problems:

1. Current spi calculus presented by different researchers (R. Focardi and M. Maffei, 2004, Y. Gu et al, 2005, A. Tiu and J. Dawson, 2010) is not suitable for open environment.

2. Current researches in spi calculus are incapable of evaluating the actions in the running processes and couldn't make suitable decisions with a changeable behavior of the open environment. Their works didn't have any ability for deciding the validity of protocol processes running in open environment that needs for automated function for evaluating each action taken by the processes.

3. Some researchers built an evaluation function (H. Al-Refai, 2009, J. Borgstrom, 2009), but they didn't overcome the complicated message structures that involve hash functions, timestamp and public key cryptography which are mostly needed for open environment in applicable protocols. They did not support tuple of messages (series of messages) such case of sending multi-messages in a single transaction. They did not evaluate the freshness of generated keys and names.

4. Complexity of the nested encryption/ decryption operations.

1.8 *Objective of the Research:*

Our main objective is to build an improved version of Spi calculus, Ph-spi calculus, to solve all mentioned problems.

I. The needs from this improved version of Spi calculus to give the ability for evaluating each action in the running processes and making suitable decision to be more suitable with open environment.

II. We need this improved version of calculus to be a step toward automated symbolic language with an evaluation functions.

III. We need this improved version of calculus to solve the problem of tuple (series) of messages we built partial map function then apply the evolution function on it.

IV. Also, we need this improved version of calculus to include special functions that mostly used in real cryptographic protocols such as timestamp, hash function, and digital signature.

1.9 *Research Methodology:*

- Based on most of the related work in reasoning on cryptographic protocols.

- Then, we apply the current spi calculus on one of the real protocols, to define the lack of current versions of the calculus spi to be appropriate for such real protocols.

- Then, after we introduce a needed parameters and operators to build an improved version of the spi calculus to make it a step toward automated symbolic language with an evaluation function. Also, to give this calculus the ability for evaluating each action in the running processes and making suitable decision to be more suitable with open environment.

- Finally, we used this improved calculus to analyze and reasoning on cryptographic protocol intended to prove the soundness and correctness using case study.

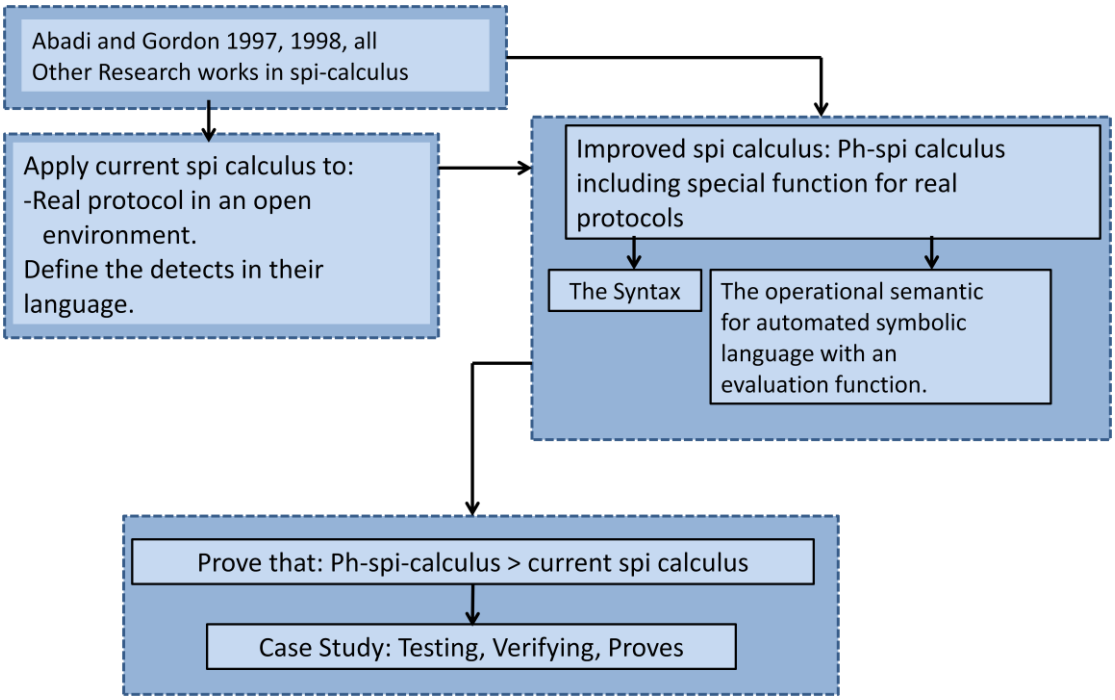

Figure 1.9.1 Research Methodology

1.10 *Overview of the Thesis:*

This thesis organized as follows:

Chapter II will cover research undertaken in the related literature with some analysis and comparisons and at the end we will give a comparisons table between them. In Chapter III we will introduce our enhanced calculus called Ph-spi calculus with the evaluation function. Chapter IV will present an example and a case study to prove the functionality of this enhanced calculus. And in the last Chapter V, we will conclude this thesis by summarizing the results and findings from the test collections. It also provides suggestions for further research work.

Chapter II:

LITERATURE REVIEW

2.1 *Introduction*

Most of research works in formal methods has been focused on analysis of discrete system. This is understandable with regard to the fact that many security problems in protocol analysis can be formulated in terms of the properties of a discrete system. Secure data exchanges between several principals are often subject to attack by an intruder. Such an intruder can do any sequence of a finite set of operations such as intercepting data, concatenating and non-concatenating data, encrypting and decrypting data, and so forth. However, some sort of discrete formal analysis can enable us to avoid attacks especially since attacks are often non-intuitive, exploitations of protocol. For this reason, formal methods are useful for the analysis of the security of cryptographic protocols. They allow one to do a thorough analysis of the different paths intruders can take, and to specify precisely the environmental assumptions that have been made.

In 1997, the spi calculus introduced as a new approach to formal cryptographic protocol analysis, the spi calculus proposed as an extension of the π-calculus (R. Milner. 1992). The spi calculus is a process calculus designed for the description and formal verification of cryptographic protocols. The spi calculus enables us to consider cryptographic issues in more detail. In spi calculus, protocols represented as processes and state their security properties in terms of protocol equivalence.

In this chapter, we will cover research undertaken in the areas related to the scope of the framework of this thesis and reviewed in the previous chapter.

2.2 *Spi Calculus Literature*

The spi calculus presented in (M. Abadi and A. Gordon 1998) as an extension of the π-calculus used to analyze and formally verify of cryptographic protocol. They present simple and poor language where restriction and scope extrusion play a central role on this version of spi calculus, so their work can be applied only on simple case study protocol.

Many researchers tried to enhance spi calculus by made it applicable to specific techniques or protocols. Others tried to solve the problems of the language requirements to be suited for use in an open environment. The following is summary of the most recent work:

- Gordon and Jeffrey introduced an enhanced version of spi-calculus (A. Gordon and A. Jeffrey 2003), they used type and effect system to prove the authenticity properties for cryptographic protocols based on asymmetric cryptography, the idea behind their work is to identify formally via subtyping separately the notions of public and tainted types, this identification formalize the way nonces increase the degree of trust in data and support different styles of nonce handshake via challenge/ response types. However, their work suffered from several limitations; such that the authors define a type for the nested encryption operation, this type cannot model some form of nested encryption; such as "sign-then-encrypt" or "encrypt-then-sign". Also, their model did not solve the problem of input transition; they considered any opponent to be untrusted and did not include some of cryptographic operations such as hash function, digital signature and timestamp. Also, the authors assume that cryptographic algorithms provide perfect integrity and confidentially properties this assumption makes the model incapable to detect key-compromise attacks.

- (C.Bodei et al. 2003), the authors presented LYSA calculus as an enhanced version of the spi calculus that used one global communication channel where

all the processes had access to this channel to eliminate any unjustified protection provided by the restricted channels. On the other hand, LYSA calculus only supported symmetric cryptography protocols and its syntax did not support other cryptographic function such, asymmetric cryptography, hash function and timestamp. Also, the authors had taken in their mined perfect view of cryptography so, their language cannot detect key-compromise attacks.

- (R. Focardi and M. Maffei, 2004), the authors introduced to enhance spi calculus, the authors proposed ρ-spi calculus as a combining of the *original Spi calculus* (M. Abadi and A. Gordon, 1998) with some feature of *LYSA-calculus* (C.Bodei et al, 2003). The authors added tags to the calculus to index the message exchange, and provide primitive for declaring process identities and long-term keys. On the other hand, the authors argue that the analysis of flawed protocols in ρ-spi calculus failed when validating suggesting possible attacks. Also, the syntax of the ρ-spi calculus did not include some basic cryptographic operators that used in most of real cryptographic protocols such as; hash function, digital signature, and message digest.

- (Y. Gu et al, 2005), the authors introduced a new version of the spi calculus, called SPC calculus as an executable model for description and analysis of the security protocols. Security properties like secrecy and authenticity can be formulated as equations, where equality is interpreted as bisimulation equivalence on abstractions. However, their works suffers of set of problems concerning the structure of output messages and they argue that their work is simple and do not consider the active environment possesses infinite messages. Also, their work is not ready to take into account the symbolic method.

- (C. Haack A. Jeffrey, 2006a), the authors presented Pattern-matching spi calculus, by using pattern-matching as a primitive their language is capable of describing complex data dependencies, their work considered one way to solve

some of the problem mentioned in Chapter I, but the using of type system caused a huge complexity to their language, this made pattern-matching spi calculus inapplicable on real protocols.

- (C. Haack and A. Jeffrey, 2006b), Haack and Jeffrey proposed another work to enhance the spi calculus, the authors presented the timed-spi calculus their work gave the ability of reading global clock by adding timestamp to a simple typed spi calculus for detecting key-compromise attacks. But, their model cannot distinguish between the times needed for the attacker to crack short term keys and the amount of time that takes to timeout, the authors assumed that the timeout is less than the time needed for the intruder can crack this key. Also, their work suffer from several limitation, where timed spi calculus allows only representations of symmetric encryption and it did not support other cryptographic techniques such as public key, hash function, message digest and digital signature.

- (J. Borgstrom, 2009), the author introduce an enhanced version of spi calculus sound with respect to concrete hedged bisimilarity, also, he introduced a smooth extension of the message algebra of the spi calculus, treating complex keys and public-key cryptography in a uniform fashion. On the other hand, his work did not solve the problem of complexity of composite messages and the way of applying encryption/decryption for these messages in a nested fashion which is needed for most transaction in real protocol in open environment and he didn't include validation function that validate all the processes in the protocol. Also, he argues that the issues of finding appropriate decompositions and deciding symbolic consistency still remain.

- (H. Al-Refai, 2009), the author presented Spi-$_{H}$-calculus as an extension of the spi calculus, and he had introduced evaluation function to validate the actions on the running processes. The Spi-$_{H}$-calculus was built to be compatible with evade

bisimulation as enhancement of framed bisimulation. However, his model could not be applied on real protocols such as e-commerce protocols; he did not include all operators needed for such protocols such as timestamp, hash function, digital signature and asymmetric key cryptosystem. Finally, the messages in the Spi-$_H$-calculus are not structured messages to be same as in real protocols that have a tuple of messages.

- (A. Tiu and J. Dawson, 2010), the authors consider a version of the spi calculus where the message language allows only representations of symmetric encryption and pairing. They argue that their work and proof methods used to establish the correctness of those procedures can be extended uprightly to cover more complicated message structures which involve simple constructor/destructor languages such as natural numbers, hash functions and public key cryptography. Also, their work did not coincide with open bisimulation and they argue that: their decision procedure is complex and cannot fit with dynamic change in an open environment. Also, their work is implemented just to fit with restricted case study protocols (not real implemented protocols) and to be used in bisimulation techniques for finite spi processes.

2.3 *Conclusion*

Many researchers (R. Focardi M. Maffei, 2004, C. Haack A. Jeffrey, 2006a, 2006b, J. Borgstrom, 2009, A. Tiu and J. Dawson, 2010) had been attempted to enhance spi calculus based on (M. Abadi and A. Gordon 1998) for the purpose of analyzing and reasoning of cryptographic protocol for achieving an optimized level of security.

Based on our set of objectives, although analysis of related work had been conducted. *Table 2.3.1* shows the link between the current research and the statement of the problems as discussed in the first chapter.

Table 2.3.1: Related Work

Problems / Pre. Research	Problem-1	Problem-2	Problem-3	Problem-4
A. Gordon and A. Jeffrey 2003	X	X	X	
C. Bodei et al. 2003	X	X	X	X
R. Focardi and M. Maffei, 2004		X	X	X
Y. Gu et al, 2005	X	X	X	X
C. Haack A. Jeffrey, 2006a	X	X		
C. Haack A. Jeffrey, 2006b	X	X	X	
J. Borgstrom, 2009		X	X	X
H. Al-Refai, 2009	X		X	X
A. Tiu and J. Dawson, 2010	X	X	X	X
T. Kahsai, 2006	X	X	X	X

Chapter III:

CONTRIBUTION:

THE Ph-SPI CALCULUS

3.1 *Introduction*

Spi calculus was introduced in (M. Abadi and A. Gordon. 1998) as an approach for description and analysis of cryptographic protocols. It is an extension of π-calculus (Milner 1992).

Such spi calculus provides more detailed descriptions of cryptographic protocols than π-calculus. While π-calculus enables the representation of channels, spi calculus also enables the representation of the channel implementations in terms of cryptography. So, the main difference between π-calculus and spi calculus is that the latter includes cryptographic primitives such as *encryption* and *decryption*.

In spi calculus, the protocols are stated as processes and their properties are proved using notions of protocols equivalence, (Abadi and Gordon 1998). For instance, we can say that a protocol keeps a piece of data *X* secret by stating that the protocol with *X* is equivalent to the protocol with X', for every X'. Here, equivalence means equivalence in the eyes of the environment. The environment can interact with the protocol, attempting to create confusion between different messages or sessions. This definition of equivalence yields the desired properties for most of the security applications.

In this chapter, we will introduce an improved version of spi calculus called Ph-spi calculus, as a step toward automated symbolic language with evaluation functions. This function is capable for making suitable decisions to handle the open environment as well as evaluating and validating every action in protocol processes for proving the main security properties as authentication and confidentiality.

Here, Ph-spi calculus is ready to be used for real protocols; it includes all operators needed for such protocols such as timestamp, hash function, digital signature and asymmetric key cryptosystem. The message is structured to be same as in real protocols that have a tuple of messages.

3.2 *The Ph-Spi Calculus*

This section gives the syntax and operational semantics of the Ph-Spi calculus to improve the original works of (M. Abadi and A. Gordon. 1998) followed by the work of (H. Al-Refai. 2009) that includes an evaluation function.

3.2.1 *Syntax*

This section introduces the language of Ph-spi calculus. Names and operators are the basic constructs the syntax of Ph-spi calculus. Other structures are built based on them. Protocol model, is representative structure of Ph-Spi-calculus. In this structure messages, expressions, logical formula and processes define the attributes needed to express all the objects and the activities driven in establishing protocols. *Tables 3.1.a* and *3.1.b* summarize the syntax of Ph-spi calculus as given in (H. Al-Refai. 2009). The bolded expressions, guards and processes are newly added.

Names $\mathcal{N}$ range over communication channels, data (variables or clear texts and a message) or keys. Along the declaration, names are used alternatively through the syntax regardless of their representations. Any name would be tagged to the type by its appearance order.

Notation 1: We reserve the lower case letters *a, b, ch* to denote channels and *k, l* to denote keys and *m, n* to denote messages.

TABLE 3.1.a: The Syntax of the Calculus

$a,b,c\ldots,k,l,m,n\ldots,x,y,z$		*names* N
$\zeta,\eta ::= a$	$\mid Enc^{+}_{\eta}(\zeta_i) \mid Dec^{+}_{\eta}(\zeta_i) \mid Dec^{-}_{\eta}(\zeta_i) \mid Enc^{-}_{\eta}(\zeta_i)$	
	$\mid \langle \zeta_1,\zeta_2 \rangle \mid \pi_1(\zeta) \mid \pi_2(\zeta) \mid \#(\zeta_i)$	expressions $\Im$
$M,N ::= a$	$\mid \langle M_1,M_2 \rangle \mid Enc_k\{M\}$	*messages* M
$\phi,\psi ::= tt$	$\mid \phi \wedge \phi \mid \neg\phi$	*guards* Φ
	$\mid [\zeta = \eta]$	*(equality)*
	$\mid [\zeta : \mathrm{N}]$	*(is a free name)*
	$\mid \lvert[\zeta : N]\rvert$	***(is a bound name)***
	$\mid [\zeta : \mathrm{M}]$	*(is a decomposability)*
	$\mid$ *let* $z=\zeta$ *in* ϕ	*(decryption)*

Expressions are those descriptions that are obtained by applying encryption, decryption, paring and projection operators to names and ciphertext. For example, the expression ζ is a plaintext, when it is encrypted using the value η as a key, the overall actions would be given by $Enc_{\eta}(\zeta)$, yielding that ζ as a plaintext is encrypted under the key η, conversely the decryption action $Dec_{\eta}\{\zeta\}$ stands to decrypt the ciphertext using the value of η as a key. We assume that a key should be a name and the encryption action uses shared key in a simple and nested modes.

TABLE 3.1.b The Syntax of the Calculus

$P,Q,R ::=$		*processes P*
	0	*(null)*
|	$\eta(x_1,.....,x_n).P$	***(input prefix)***
|	$\bar{\eta}\langle\zeta_1,.....,\zeta_n\rangle.P$	***(output prefix)***
|	$P \mid Q$	*(parallel compositon)*
|	$P+Q$	*(choice)*
|	$!P$	*(replication)*
|	$(\nu n)P$	*(restriction)*
|	ϕP	*(booleanguard)*
|	$let\ z=\zeta\ in\ P$	*(encryption/ decryption)*
|	***let η : P***	***(Timestamp)***

Definition 3.2.1.1: Let *P* represent the running process, *Q* for any other process running in the environment, and e represent the knowledge of the environment then the set of all free names known to the environment is defined by Γ, such that, $\forall\ (fn(P) \cup fn(Q) \cup fn(e) \in \Gamma$.

Definition 3.2.1.2: Let *P* represent the running process, *Q* for any other process running in the environment, and e represent the knowledge of the environment then the set of all free variable known to the environment is defined by Γ, such that, $\forall\ (fv(P) \cup fv(Q) \cup fv(e) \in\ \Gamma$.

Logical formulae generalize the usual equality operator of the π-calculus by conjunction and negation. Moreover we introduce some new predicates let η *:P* , $[\zeta : \mathcal{N}]$, $[\zeta : \mathcal{M}]$ and $\|[\zeta : \mathcal{N}]$. The predicate *let* η*:P* means η time is evaluated as a valid current time of process *P*, and the predicate $[\zeta : \mathcal{N}]$ which tests for the format of the argument ζ , whether it evaluates to a name or not, and the predicate $[\zeta : \mathcal{M}]$ which test for the argument ζ , whether it evaluates to a compound ciphertext or not, and with "*Let"* construct that binds the value of some expression ζ to a name *z*, and $\|[\zeta : \mathcal{N}]$ which tests for the format of the argument ζ , if it is bounded in the process *P and not belong to names in* any other processes in the environment, such as $\zeta \notin (\ fn(P) \cup bn(Q) \cup \Gamma$.

There are other expressions denote hash function and the encryption/ decryption using public and private keys:

- Hash function: $\#(\zeta_i)$ hashing of ζ_i. The hashing function assumed to be perfect and noninvertible.

- $Enc_k^+(\zeta_i)$: Encrypt the structured message ζ_i using a public key k^+.

- $Enc_k^-(\zeta_i)$: Encrypt the structured message ζ_i using a secret key k^-.

- $Enc_k^+\{\zeta_i\}$: Encrypt the structured message (ζ_i) that signed digitally by public key k^+.

Definition 3.2.1.3: A finite set of terms, $T = \{t_1, t_2, \ldots, t_n\}$. (H. Al-Refai, 2009).

Following the work of (H. Al-Refai. 2009), the difference to the old version of spi calculus is that we assumed the set *T* to be defined, so that we can associate each state to a term *t*. In order to define the notion of *state* we have to introduce the definition of finite multisets.

Definition 3.2.1.4: A f*inite multiset* ϖ over a set *L* is a map $\varpi: L \rightarrow M$ such that $\varpi^{-1}(M_1)$ is finite. We define the following operations on finite multisets: (H. Al-Refai, 2009).

(a) The *difference* of the multisets ϖ and ϖ' is the multiset $\varpi|\varpi'$ where $(\varpi|\varpi')(l) = max\,(0,\ \varpi(l) - \varpi'(l))$;

(b) The *union* of two multisets ϖ and ϖ' is the multiset $\varpi \cup \varpi'$ where $(\varpi \cup \varpi')(l) = \varpi(l) + \varpi'(l)$;

(c) We say that $l > \varpi$ *iff* $\varpi(l) > 0$.

We define formally our notion of *state* as follows:

Definition 3.2.1.5: We define a *state,* $\sigma = \{\sigma_t\}_{t \in T}$, to be a family of multisets indexed by the terms, where each σ_t represents the local state of the term *t*. We denote the set of all states by $\sum$ from (H. Al-Refai. 2009).

A first approach to the definition of state would be a family of sets. If we consider each σ_t as a set, we were restricting the possibility of a principal to have many copies of the same term. We want to deal with the possibility of existing several inputs of the terms and verify that our system has the desired properties for input transitions.

Definition 3.2.1.6: The participant γ and λ only accepts the fresh key $\mu_{\gamma\lambda}$ generated by *S* as server.

If two processes can be made equal by conflict-free renaming of bound names then they are alpha-equivalent. Substitutions σ are mappings $\{\zeta/x\}$ from names *x* to messages ζ, following the usual assumption that name-capture is avoided through implicit alpha conversion. Substitutions are applied to processes, expressions and guards very simply as for example, $P\{\zeta/x\}$ replace all free occurrence of *x* in *P* by ζ, possibly renaming bound names in *P* avoiding name capture.

Definition 3.2.1.7: A substitution σ is a finite partial map from the set of names *N* to the set of messages *M*; $\{M_i/x_i\}$.

The definition shows the effect of applying a substitution σ to a process *P*. This is essentially to replace each free occurrence of each name (*i.e.; x*) in *P* by $\sigma(x_i) = M_i$, for some x_i and M_i. The mapping must, however, be done in such a way that unintended capture of names by binders is avoided. Substitutions are applied to processes, expressions and guards in straightly, i.e. $P\{M_i/x_i\}$ replace all free occurrence of x_i in *P* by M_i, possibly renaming bound names in *P* avoiding name capture.

Definition 3.2.1.8: (σ-convertibility)

1. If the name *w* does not occur in the process *P*, then *P{w/z}* is the process obtained by replacing each free occurrences of *z* in *P* by *w*.
2. A change of bound names in a process *P* is the replacement of a subterm *x(z).Q* of *P* by *x(w).Q{w/z}*, or the replacement of subterm *vz Q* of *P* by *vw Q{w/z}*, where in each case w does not occur in *Q*.
3. Processes P and Q are σ-convertible, *P = Q*, if *Q* can be obtained from *P* by a finite number of changes of bound names.

Remark: we have use the symbol '=' for σ-convertibility because we intend to identify σ-convertibility processes. In our spi calculus, the entities of interest are the equivalence classes of processes modulo σ-convertibility.

The domain and co-domain of σ are denoted as *dom*(σ) and *range*(σ) respectively .Let $n(\sigma)= \quad dom(\sigma) \cup \left(\bigcup_{M \in range(\sigma)} n(M) \right)$ When a tuple of distinct names $\vec{x} = (x_1, x_2, ..., x_n)$ and a tuple of messages $\vec{M} = (M_1, M_2, ..., M_n)$ are given , the substitutions mapping of each x_i to M_i will be convenient. Usually, a tuple is a set of its component. $\sigma\left[\vec{M}/\vec{x}\right]$ is the substitution σ' which represents union of σ and $\left[\vec{M}/\vec{x}\right]$. Such case is referred to as σ' extends σ.

Processes are diverse forms of processes that have a distinct function for each. A process could be built using a set of operators that include standard *π-calculus* (Milner et al. 1992) and four new ones; symmetric and asymmetric cryptosystem, Boolean guards, timestamp and structured message. However, process forms are used to explain the following:

- **0**, is a null process that dose nothing.

- An input process; $\eta(\zeta_1, ..., \zeta_n).P$ represents input of a generic message *x* along the channel η : the only useful case is when η is a name , otherwise the whole process is stuck .

- An output process $\overline{\eta}(\zeta_1, ..., \zeta_n).P$ represents out of ζ along the channel η .The only useful case is when η is a name and ζ is a message, otherwise the whole process is stuck.

- Non-deterministic choice *P* + *Q*: can behave either as *P* or *Q*; the choice might either be triggereazd by the environment, or by internal computations of *P* or *Q*.

- Parallel composition *P* | *Q*; is the parallel execution of *P* and *Q*.

- Restriction *(νn)P*: creates a new name *a* which is only known to *P*.

- Replication *!P* behaves like many unbounded copies of *P* running in parallel, i.e. *P* | *P* | *P* | **….** .

- Boolean Guard ϕP behaves like P if the formula is logically true, otherwise is stuck.

- Encryption/Decryption *let* $z = (\zeta_1, ..., \zeta_2)$ *in* P: referring to definitions 3.2.2.6, 3.2.1.6 and 3.2.1.7 attempts evaluation of ζ: if the evaluation succeeds, the result bound to z within P, in other words, the process will attempts to encrypt/decrypt z with the key η. If z has the form $\{M_1, ..., M_n\}_k$, then the process behaves as the process P, where each ζ_i has been replaced by M_i, ie. as the process $P\{M_1/x_1, ……., M_n/x_n\}$.otherwise the whole process is stuck.

- Timestamp: the process let $T:P$ means T time is evaluated as a valid current time of process P.

Usual calculus abbreviate $(\nu\, a)(\nu\, b)\ P$ into $(\nu\, a, b)\ P$, and $\overline{M}(N).0$ into $\overline{M}(N)$. In the Ph-spi calculus, processes are identified up to renaming of bound names. Bound names are the entities that are enclosed within a process definition P, and not those that have explicitly been tagged with any other outside the process. So the closed process will then be defined as the process that has no free variables; *Proc is used to define a set of closed processes.* For that we will propose in our operational semantics an evaluation function to validate if the bound names did not disclosed with an environment out of the process definition.

Let *fn(P)* denote the set of free names in P and *fv(P)* is the set of free variables in P , and alpha-equivalence arises as expected , *n(P) is fn(P)* $\cup$ *bn(P).* In this context, similar notations are used for formulae, expressions and messages.

3.2.2 *Operational Semantics*

In (Abadi and Gordon 1998), there are two operational semantics presented for spi-calculus - the *reaction relation* and the *commitment relation*. The *reaction relation* is an adaptation of a similar idea introduced by Milner. The definition of reaction is rather elegant, but not convenient for proofs (because it relies on an auxiliary notion of structural equivalence). Therefore, an alternative characterization of reaction is provided defining the *commitment relation*, in style of (R. Milner 1999). We will present both as in (Abadi and Gordon 1998). For more details see appendix A.1.

However, to provide a calculus to be toward automation, this needs for two main evaluation functions are used (H. Al-Refai. 2009). The first is utilized for expressions while the second is used for Boolean Guards. These two evaluations are denoted as follows:

- For an expression $\overline{\overline{\cdot}} : \varepsilon \to M \cup \{\perp\}$ *where* $\perp$ *is a distinct symbol* $(\perp \notin M)$, where (.) is represent an expression.

- For an evaluation of a Guard $\underline{\underline{\cdot}} : \Phi \to \{tt, ff\}$, *is defined by induction on* Φ. where is represent guard.

The evaluation function is defined recursively according to tables 3.2, and 3.3. We have introduced one new expression { $\overline{\overline{\#(\zeta)}}$ } and four Boolean guards { $\underline{\underline{let\ \eta : P}}$, $\underline{\underline{let\ z_i = \zeta_i\ in\ \phi}}$, $\underline{\underline{|[\ \zeta : N\ ]|}}$ and $\underline{\underline{[\ \zeta = \eta\ ]}}$ } evaluations. In these two tables, it is obvious that expression evaluations rely on the implementation of *let* and *guard* only. Hence, the decryption is bounded along this evaluation scheme.

The evaluation function performs a tracing to every process action in the protocol for deciding either to proceed or terminate that process. The role of each function is indicated in the tables. For example:

$$\overline{\overline{Enc_{\zeta}(\eta)}} = \begin{cases} Enc_k(M) & \text{if } \overline{\overline{\eta}} = M \in \mathcal{M} \text{ and } \overline{\overline{\zeta}} = k \in \mathcal{N} \\ \perp & \text{otherwise.} \end{cases}$$

Evaluated as: the plaintext η is evaluated as a message and belong to the defined set of messages $\mathcal{M}$ and the key ζ evaluated as a name and belong to the defined set of names $\mathcal{N}$, if these conditions success the operation evaluated to be valid otherwise terminate the process.

TABLE 3.2: Expression evaluation in the Ph-spi calculus

$\overline{\overline{a}}$	$= a$
$\overline{\overline{Enc_{\zeta}(\eta)}}$	$= \begin{cases} Enc_k(M) & \text{if } \overline{\overline{\eta}} = M \in \mathcal{M} \text{ and } \overline{\overline{\zeta}} = k \in \mathcal{N} \\ \perp & \text{otherwise.} \end{cases}$
$\overline{\overline{Dec_{\zeta}(\eta)}}$	$= \begin{cases} M & \text{if } \overline{\overline{\eta}} = Enc_k(M) \in \mathcal{M} \text{ and } \overline{\overline{\zeta}} = k \in \mathcal{N} \\ \perp & \text{otherwise} \end{cases}$
$\overline{\overline{\langle \zeta_1, \zeta_2 \rangle}}$	$= \begin{cases} \langle M_1, M_2 \rangle & \text{if } \overline{\overline{\zeta_1}} = M_1 \text{ and } \overline{\overline{\zeta_2}} = M_2, \\ & \text{for some } M_1 \text{ and } M_2 \in \mathcal{M} \\ \perp & \text{otherwise} \end{cases}$
for $i=1,2$, $\overline{\overline{\pi_i(\zeta)}}$	$= \begin{cases} M_i & \text{if } \overline{\overline{\zeta}} = \langle M_1, M_2 \rangle, \text{for some } M_1 \text{ and } M_2 \in \mathcal{M} \\ \perp & \text{otherwise} \end{cases}$
$\overline{\overline{\#(\zeta_i)}}$	$= \begin{cases} \#(M_i) & \text{if } \zeta_i = M_i \in \mathcal{M} \text{ and } \#(\zeta_i) \neq \#(\zeta_i)^{-1} \\ \perp & \text{otherwise} \end{cases}$

TABLE 3.3 Boolean Guard evaluation in the Ph-spi calculus

$\underline{\underline{tt}}$	$=$	tt
$\underline{\underline{\phi \wedge \psi}}$	$=$	$\underline{\underline{\phi}} \wedge \underline{\underline{\psi}}$
$\underline{\underline{\neg \psi}}$	$=$	$\neg \underline{\underline{\psi}}$
$\underline{\underline{let\ z_i = \zeta_i\ in\ \phi}}$	$=$	$\begin{cases} \underline{\underline{\phi\left\{M_i / x_i\right\}}} & if\ \underline{\underline{\zeta}} = M \in \mathcal{M} \\ \quad and\ x \in \mathcal{N} \notin \left(fn(.) \cup bn(Q) \cup \Gamma\right) & \\ ff & otherwise \end{cases}$
$\underline{\underline{let\ \eta : P}}$	$=$	$\begin{cases} Tt & if\ \underline{\eta} \in \mathcal{N} \notin (fn(P) \cup bn(Q) \cup \Gamma) \\ ff & otherwise \end{cases}$
$\underline{\underline{[\zeta : \mathrm{N}]}}$	$=$	$\begin{cases} tt & if\ \zeta \in \mathcal{N} \\ ff & otherwise \end{cases}$
$\underline{\underline{\lvert[\zeta : \mathrm{N}]\rvert}}$	$=$	$\begin{cases} tt & if\ \zeta \in \mathcal{N}\ \text{and}\ \zeta \notin \left(fn(.) \cup bn(Q) \cup \Gamma\right) \\ ff & otherwise \end{cases}$
$\underline{\underline{[\zeta : \mathrm{M}]}}$	$=$	$\begin{cases} tt & if\ \underline{\underline{\zeta}} \neq \perp \\ ff & otherwise \end{cases}$
$\underline{\underline{[\zeta = \eta]}}$	$=$	$\begin{cases} tt & if\ \underline{\underline{\underline{\zeta}}} = \underline{\underline{\underline{\eta}}} \in \mathcal{M} \\ ff & otherwise \end{cases}$

Table 3.4 shows, most of the operational semantics of the Ph-Spi-calculus used in our work is similar to the work of (H. Al-Refai, 2009). *Let* and guard items are used as primitive rules driven to decrypt messages. A process ΦP behaves like P provided

that Φ evaluates to true, otherwise, ΦP is stuck. A process *let* $z = (\zeta_1, ..., \zeta_2)$ *in* P provided that the evaluation of ζ succeeds; otherwise, *let* $z = (\zeta_1, ..., \zeta_2)$ *in* P is stuck. If z has the form $\{M_1, ..., M_n\}_k$, then the process behaves as the process P, where each ζ_i has been evaluated by M_i, ie. as the process $P\{M_1/x_1, ……., M_n/x_n\}$.otherwise the whole process is stuck.

Rule (E-OUT) details the case when the environment receives a message M and updates its knowledge accordingly, and for the sake of a transition to occur, all channels are supposed to be well announced to the environment.

Rule (E-INP) details the case, when the environment sends a message M to the process. Message m is not arbitrary and the expression ζ describes how this message is built out of the environment and of the names $\vec{b}$ to define M. Creation of new names $\vec{b}$ is recorded by $\left[\vec{b}/\vec{x}\right]$, and in this case a must belong to the knowledge of the environment to explain its announcement.

In table 3.4, the evaluation function is built in the operational semantic to strengthens the validation rules of the language. For more details see appendix A.1.

TABLE 3.4 The Operational Semantics of the Ph-Spi calculus

$$(OUT)\ \frac{[\eta : \mathrm{N}] = a \qquad [\zeta : \mathrm{M}] = M}{\overline{\eta}\langle\zeta\rangle.P \xrightarrow{\overline{a}M} P} \qquad (INP)\ \frac{[\eta : \mathrm{N}] = a}{\eta(x).P \xrightarrow{aM} P\{M/x\}}$$

$$(SUM)\ \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$$

$$(REP)\ \frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'} \qquad (ALPH)\ \frac{Q \xrightarrow{\mu} Q' \quad P \equiv_\alpha Q}{P \xrightarrow{\mu} Q'}$$

$$(PAR)\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \text{if } \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$(RES)\ \frac{P \xrightarrow{\mu} P'}{(\nu c)P \xrightarrow{\mu} (\nu c)P'}\ \text{if } c \notin n(\mu)$$

$$(OPEN)\ \frac{P \xrightarrow{(\upsilon\, \overline{\mu})\overline{\eta}(\zeta)} P'}{(\nu c)P \xrightarrow{(\nu c\overline{\mu})\overline{\eta}(\zeta)} P'}\ \text{if } \overline{\overline{c}} = \mathrm{n}(M) \ni c \notin \{\eta, \overline{\mu}\}$$

$$(COM)\ \frac{P \xrightarrow{a\,M} P' \qquad Q \xrightarrow{(\nu\vec{b})\overline{a}\,M} Q'}{P \mid Q \xrightarrow{T} (\nu\vec{b})(P'\left[M/x\right] \mid Q')}\ \text{if } \{\vec{b}\} \cap \mathrm{fn}(P) = \emptyset$$

$$(GUARD)\ \frac{P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'}\ \text{if } \overline{\overline{\phi}} = tt \qquad (LET)\ \frac{P\{\overline{\overline{\zeta}}/z\} \xrightarrow{\mu} P'}{let\ z = \zeta\ \mathrm{in}\ P \xrightarrow{\mu} P'}\ \text{if } \overline{\overline{\zeta}} \neq \perp$$

3.3 Conclusion:

In this chapter, we have introduced an improved calculus that called Ph-spi calculus it is a symbolic language that to be toward an automated language. Therefore, we have built an evaluation function to validate the process actions in each step of the protocol. In the next chapter, this improved calculus will be used to formulate a simple example that had been used by many researchers such as (M. Abadi A. Gordon. 1998), then a case study will be given, using kerberos protocol.

Chapter IV:

PROOFS AND CASE STUDY

4.1 *Introduction*

In previous chapter, we have presented the Ph-spi calculus that supports real protocols such as e-commerce protocols running within open environment. Our calculus includes an evaluation function that used for validating process actions and all operators needed such as timestamp, hash function, digital signature and asymmetric key cryptosystem. The message is structured to be same as in real protocols that have a tuple of messages.

In this chapter, we will apply our proposed Ph-spi calculus to an example that is used in many research works to prove the soundness of our proposed calculus (R. Focardi and M. Maffei, 2004, H. Al-Refai, 2009, A. Tiu and J. Dawson, 2010). Then, in the next section we will apply our proposed calculus on kerberos protocol with a tuple of messages as a case study.

Example 4.1.1:

The first example with cryptographic is extremely basic. In this example, we consider two principals *A* and *B* that share a key μ_{AB}; in addition, we assume there is a public unsecure channel c_{AB} that *A* and *B* can use for communication. The protocol simply depicts that *A* sends a message *M* under the key μ_{AB} to *B*, on a channel η.

μ_{AB} shared key between A and B.

ζ Message ∈ $\mathcal{M}$.

A → B : $Enc_{\mu_{AB}}\{\zeta\}$ on channel η

In Ph-spi calculus:

$$A(\zeta) ::= \bar{\eta}\,(\mathrm{Enc}_{\mu_{AB}}\{\zeta\})$$

$$B \quad ::= \eta(x).\, let\ z = \ \zeta\ \ in\ P$$

$$Inst\ (\zeta) ::= (v\mu_{AB})\ (A(\zeta)|B)$$

In this example, the freshness of the generated key $(v\ \mu_{AB})$ is guaranteed by the evolution μ to be bounded in process P and not disclosed out of the scope of P. this can be by the use of the evaluation function for bound names as:

$\underline{|\{\mu : N\}|.\, P}$ to means that μ is a name bounded by P and $\mu \notin (fn \cup bn(.) \cup \Gamma)$.

Now, after proving the freshness of the key μ_{AB}, we have to evaluate the encryption process $Enc_{\mu_{AB}}\{\zeta\}$ doing that using the evaluation at the encryption defined in our evaluation function, such as : $\overline{\overline{Enc_{\mu}(\zeta)}}$ the evaluation will check if $\overline{\overline{\zeta}}$ evaluated as $Enc_k(M) \in \mathcal{M}$ to mean that ζ should be constructed as a message and the key μ evaluated as $\bar{\bar{\mu}} = k \in \mathcal{N}$ means that the key should be a name, otherwise the whole process will stack $\perp$.

Form the above example, and by comparing it with the same one in abadi work (M. Abadi, A. Gordon 1998), we found that, by using our version of spi calculus we can solve some of the earlier problems concerning the freshness of generated names and guarantee of the bound names to be closed in the process domain.

Note, the old versions of spi calculus use the same example, but they assume the channel is restricted to be used only by A and B.

For B attempts to evaluate ζ as: if the evaluation succeeds; the process will behave as{M/x} where $\zeta = M \in \mathcal{M}$ and $x \in \mathcal{N} \notin (fn(P) \cup bn(.) \cup \Gamma)$, where (.) for any other processes running in the open environment, then the result bound to z within P, otherwise the whole process stack $\perp$.

Now, we use the following specification.

$A(\zeta)$ ::= $\bar{\eta}\ (Enc_{\mu_{AB}}\{\zeta\})$

B_{Spec} ::= $\eta(x).let\ z = \zeta\ in\ P$

$Inst_{Spec}(\zeta)$ ::= $(v\mu_{AB})\ (A(\zeta)|B_{Spec})$

And we obtain the properties:

Authenticity: $Inst(\zeta) \cong Inst\ (\zeta')\ if\ \zeta \cong \zeta'$, for all ζ and ζ'.

For more details and proof, reader can refer to Appendix A.1.

4.2 *Case Study: The KERBEROS Protocol*

The Kerberos protocol (*Figure* 4.2.1) is a key distribution protocol between a client and an applied server which is based on Needham- Schroeder protocol. We only consider a version taken for this protocol proposed in (M. Burrow et al. 1990) and (M. Boreale et al. 2000). We consider such protocol which has a structured message that contains set of elements in a single transaction to prove one of our contributions. That is, one of the improvements for the language we have introduced in this work concern such case.

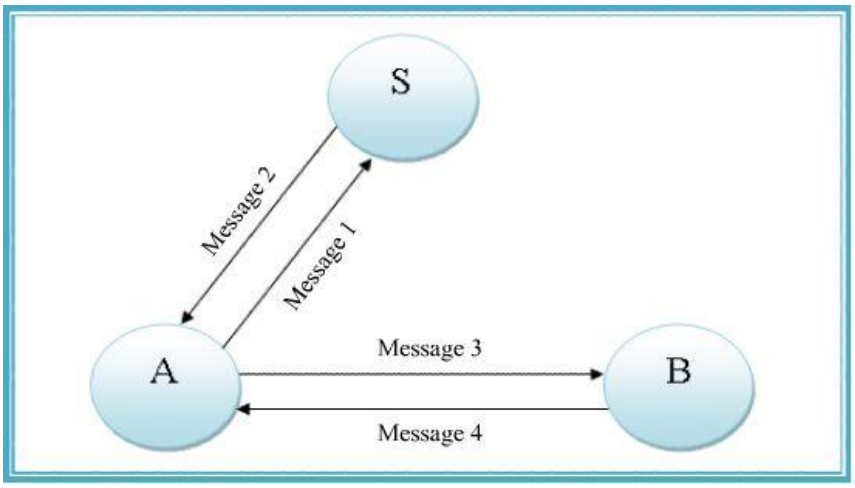

Figure 4.2.1: Kerberos protocol

The system in our case has two agents γ (initiator) and λ (responder) that each one of them share long term key with a server *S* ($\mu_{\gamma s}$ and $\mu_{\lambda S}$), where γ communicate with the server *S* to establish new connection with λ. The symbols used in our case study are shown in *Table* 4.2.1.

Table 4.2.1 Case study symbols

Symbol	Represent
γ	Initiator
λ	Responder
γ_{id}	Initiator ID
λ_{id}	Responder ID
ζ	Is a Message
μ	Is a key
ι	Is a timestamp
δ_{λ}	Responder Certificate
η	Public channel

Informally, this protocol is shown below:

Message 1 $\gamma \rightarrow S$: γ, λ

Message 2 $S \rightarrow \gamma$: $\{n_s,\ k_{\gamma\lambda}, \lambda, cert_{\lambda}\}_{k_{S\gamma}}$

Message 3 $\gamma \rightarrow \lambda$: $cert_{\lambda}, \{\gamma, n_{\gamma}\}_{K_{\gamma\lambda}}$

Message 4 $\lambda \rightarrow \gamma$: $\{n_{\gamma}\}_{K_{\gamma\lambda}}$

First of all, to implement such protocol in Ph-spi calculus we consider the run of the protocol under the hypothesis that the old session key possible has been released to the environment. So, the new generated keys should not release any of this knowledge to the environment till the running session completely finished and evaluate the knowledge gained previously by the environment for old keys in previous sessions.

In the following we present some notational shorthand, in order of convenience, which define the outputted messages that will be sent through public channels:

$$\zeta_0 : (Enc_{\mu_{\lambda S}}\{\iota_0, \mu_0, \gamma_{id}\}, \mu)$$

$$\zeta_1 : (\gamma_{id},\ \lambda_{id})$$

$$\zeta_2 : (Enc_{\mu_{\gamma S}}\{\iota_s, \mu_{\gamma\lambda}, \lambda_{id}, \delta_\lambda\})$$

$$\zeta_3 : (\delta_\lambda,\ Enc_{\mu_{\gamma\lambda}}\{\gamma_{id}, \iota_\gamma\})$$

$$\zeta_4 : (Enc_{\mu_{\gamma\lambda}}\{\iota_\gamma\})$$

$$\zeta_\lambda : (Enc_{\mu_{\lambda S}}\{\iota_s, \mu_{\gamma\lambda}, \gamma_{id}\})$$

- ζ_0 denotes the old message including the old timestamp (nonce) ι_0 and session key μ supposed to be known by the environment. δ_λ indicates that it is a new certificate awarded by the Kerberos server S to the initiator γ.

The specifications are similar to (M. Boreale et al. 2000). Firstly, we have to describe the implementation of this protocol in the Ph-spi calculus in order to formally give the specification.

$$\gamma \quad ::= \quad \bar{\eta}(\zeta_1).\eta(x_{\zeta_2}).\bar{\eta}(\zeta_3).\eta(x_{\zeta_4})$$

$$\lambda \quad ::= \quad \eta(x_{\zeta_3}).\ \bar{\eta}(\zeta_4)$$

$$S \quad ::= \quad \eta(x_{\zeta_1}).\ \bar{\eta}(\zeta_2)$$

$$Kerb \quad ::= \quad (\nu\, \mu_{\gamma S}, \mu_{\lambda S})\ (\bar{\eta}(\zeta_0) \mid (\nu\, \iota_\gamma)\, \gamma \mid \lambda \mid (\nu\, \iota_s, \mu_{\gamma\lambda}) S)$$

Where: *x* derived from the partial-map function as:

$$\sigma_1\,(x_{\zeta_1}) ::= \quad (\{\zeta_1/x_i\}) => (\{\gamma_{id}/x_1\,,\lambda_{id}/x_2\})$$

$$\sigma_2\,(x_{\zeta_2}) ::= \quad (\{\zeta_2/x_i\}) => (\{\iota_s/x_1\,,\mu_{\gamma\lambda}/\,x_2\,,\lambda_{id}/x_3\,,\delta_\lambda/x_4\})$$

$$\sigma_3\,(x_{\zeta_3}) ::= \quad (\{\zeta_3/x_i\}) => (\{\gamma_{id}/x_1\,,\iota_\gamma/x_2\})$$

$$\sigma_4\,(x_{\zeta_4}) ::= \quad (\{\zeta_4/x_i\}) => (\{\,\iota_\gamma/x_1\})$$

The evaluation function will verify the freshness of all generated nonces, as well as the encryption/decryption operations and validate the structure of messages, the bound names projection of structured messages; all of these empower the analysis and tracing of the protocol. As well as prove later in this chapter. Therefore, the specification will be followed simply for verifying authentication and secrecy as we will prove later in this chapter.

The specification of kerb is formally described as:

$$\gamma_{spec} \quad ::= \quad \bar{\eta}(\zeta_1).\eta\,(x_{\zeta_2}).\bar{\eta}\,(\zeta_3).\eta\,(x_{\zeta_4})$$

$$\lambda_{spec} \quad ::= \quad \eta\,(x_{\zeta_3}).\ \bar{\eta}\,(\zeta_4)$$

$$S_{spec} \quad ::= \quad S$$

$$Kerb_{spec} \quad ::= \quad (v\,\mu_{\gamma S},\mu_{\lambda S},\mu_{\gamma\lambda})\,(\overline{\eta}\,(\zeta_0)\mid(v\,\iota_\gamma)\,\gamma_{spec}\mid\lambda_{spec}\mid(v\,\iota_s)S\,)$$

Proof:

To prove the authentication property, the set of names should be all distinct from each other and not known by the environment. For example we can prove the validity of ζ_2, such as:

- Firstly, we evaluate the freshness nonce ι_s as a timestamp by $\underline{\underline{[\iota_s \neq \iota_0]}}$ and $\underline{\underline{|[\iota_s : N]|}}$ that means ι_s is freshly generated as a name and bounded in the running process. Such that, the $\iota_s$is valid timestamp if $\underline{\underline{\iota_s}} \in \mathcal{N}$ and $\underline{\underline{\iota_s}} \notin (fn(P) \cup bn(Q) \cup \Gamma)$ for the running process *P* and any other process *Q* in the environment.

- The same for evaluating $\underline{\underline{\delta_\lambda}}$ by $\underline{\underline{|[\delta_\lambda : N]|}}$ for $\underline{\underline{\delta_\lambda}}$ is valid if $\underline{\underline{\delta_\lambda}} \in \mathcal{N}$ and $\underline{\underline{\delta_\lambda}} \notin (fn(P) \cup bn(Q))$ for the running process *P* and any other process *Q* in the environment.

  The encryption process evaluated as $\overline{\overline{Enc_{\mu_{\gamma\lambda}}(\zeta_2)}}$ that is, it valid if $\overline{\overline{\zeta_2 = Enc_k(M) \in M}}$ and $\overline{\overline{\mu_{\gamma\lambda}}} = k \in \mathcal{N}$ and $\overline{\overline{\mu_{\gamma\lambda}}} \notin (fn(P) \cup (bn(Q) \cup fn(Q))$ for *P* is a running process and for any other process *Q* in the environment. According the ***definition 4.2.1***, and by evaluating for the input transactions x_{ζ_2} we can observe that $x_{\zeta_2} = \{x_1, x_2, x_3, x_4\}$ and each *x* of them is distinct from the others.

  Now, the evaluation of all actions in γ, λ and *S* in *kerb* can verify the authentication property for *kerb* and $kerb_{spec}$.

Definition 4.2.1: The participant γ and λ only accept the fresh key $\mu_{\gamma\lambda}$ generated by *S*.

For any case, where the evaluation function result an error that will signal to break the process $\perp$.

This will prove that *kerb* and $kerb_{spec}$ are testing equivalence ($\simeq$) which mean that the authentication property proved.

Definition 4.2.2: Secrecy: the keys $\mu_{\gamma S}, \mu_{\lambda S}$ and $\mu_{\gamma \lambda}$ are never released.

Formally, it holds that

$$kerb \mid \varepsilon \simeq kerb_{spec}$$

Where ε represent "eavesdropping", for whom passively grab or learned from the messages known to the environment.

$$\forall\ (fn(P) \cup \Gamma)/bn(P) \in \varepsilon$$

Mean that ε contains all free names and all names collected previously from all sessions possibly learned or disclosed accidently by the environment excluding the bound names generated to be closed in running process. For more details, reader can refer to Appendix A.1.

Chapter V:

CONCLUSION AND FUTURE WORK

Cryptographic protocol is a protocol that used to protect computer systems and network transactions from malicious attacks. The design of such protocols had been known to be very hard due to their complexity.

However, cryptographic protocols are subject to many subtle attacks, so many researchers attempts to develop tools to model and analyze these protocol to check if it can guaranty the security properties. Formal method proved to be a useful method for the analyzing such protocols because its allow one to do overall analysis of the different path the intruders can take, and the ability of formal analysis to reveal previously unknowing attack, and precisely specify the environmental assumptions. In this thesis we will concern on enhancement of the *spi calculus*, which is formal tool used to analyze security protocols.

Many researchers tried to enhance the original work of Abadi and Gordon (M. Abadi and A. Gordon 1998), but most of their works built over some restriction and assumption that made the language fit to specific protocols.

In this thesis, we presented an improved version of spi calculus called Ph-spi calculus to be step toward symbolic automated language with evaluation function. This function is capable for making suitable decisions to handle the open and changeability in environment behaviors as well as evaluating and validating each action in the running processes and making suitable decision to prove the main security properties as authentication and confidentiality.

The Ph-spi calculus ready to be used for real protocols such as e-commerce protocols, it includes all operators needed for such protocols such as timestamp, hash function, digital signature and asymmetric key cryptosystem. The message is structured to be same as in real protocols that have a tuple of messages.

In future work, we plan to apply the Ph-spi calculus on real protocol that based on asymmetric key. Also, we are planning to apply one of the bisimulation techniques, such as framed bisimulation, evade bisimulation … etc, to prove the soundness of our calculus.

References

A. Gordon A. Jeffrey (2003). Types and effects for asymmetric cryptographic protocols. Submission to *Journal of Computer Security*. This material is based upon work supported by the National Science Foundation under Grant No. 0208549.

A. Tiu and J. Dawson. (2010). Automating Open Bisimulation Checking for the Spi Calculus. 23rd IEEE Computer Security Foundations Symposium.

B. Blanchet, M. Abadi, and C. (2008). Fournet. Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming, 75(1):3–51.

C. Albrechts. (2006). Automatic Analysis of Recursive Cryptographic Protocols. Master Thesis.

C. A. R. Hoare. (1980). Communicating sequential processes. In R. McKeag and A. Macnaghten, editors, On the construction of programs – an advanced course, pages 229–254. Cambridge University Press, 1980.

C. Bodei M. Buchholtz P. Degano F. Nielson H. Riis Nielson. (2003). Automatic Validation of Protocol Narration. Supported in part by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies

C. Haack A. Jeffrey. (2006). Pattern-Matching Spi-Calculus. presented at the IFIP workshop on Formal Aspects of Security and Trust in Toulouse.

C. Haack and A. Jeffrey. (2006). Timed Spi calculus with Types for Secrecy and Authenticity. This material is based upon work supported by the National Science Foundation under Grant. No. 0208459.

C. V´eronique. D. St´ephanie. (2009) A method for proving observational equivalence. 22nd IEEE Computer Security Foundations Symposium, pages 266- 276, 2009.

D. Dolev, A.C. Yao. (1981). On the security of public key protocols, In Proceedings of the 22nd Symp. On Foundations of Computer Science, pages 350-357, Nashville, Tennessee, USA. IEEE Computer Society Press.

G. Zheng; X. Li; L .Li; J. Wu. (2009). Process algebra for web servers with timed-priority executing policy Proceedings – 2009 International Conference on Computational Intelligence and Software Engineering, CiSE 2009, Wuhan China.

H. Al-Refai; T. Sembok; M. Yusoff. (2004). Decidablity of Cryptographic Protocol Properties through Framed bisimulation. Proc. International conference on Informatics. 2(1): 779-796.

H. Al-Refai (2009) Evaluation Technique in the Spi-Calculus for Cryptographic Protocols. Third International Symposium on Innovation in Information Communication Technology- ISIICT.

H. Hermanns and M. Rettelbach. (1994). Syntax, Semantics, Equivalences, and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, Proc. of the 2nd Workshop on Process Algebras and Performance Modelling, Erlangen-Regensberg, July 1994. IMMD, Universitat Erlangen-Nurnberg.

H. Huttel. (2002). Deciding framed bisimulation. In Proc. 4th Int. Workshop on Verification of Infinite State Systems (INFINITY'02).

I. Fléchais. (2005). Designing Secure and Usable Systems. PhD thesis. University of London.

J. Borgstrom. (2009). A Complete Symbolic Bisimilarity for an Extended Spi Calculus. Published by Elsevier Science B. V.

L. Durante, R. Sisto, and A. Valenzano. (2003). Automatic testing equivalence verification of spi calculus specifications. ACM Transactions on Software Engineering and Methodology, 12(2):222–284.

M. Abadi A. Gordon. (1997). A calculus for cryptographic protocols: The Spi Calculus. Proc. Zurich, Switzerland Computer and Communications Security. pp. 36–47.

M. Abadi A. Gordon. (1997). Reasoning about cryptographic protocols in the Spi Calculus. Proc. of the CONCUR'97, 8th International Conference on Concurrency Theory, 1243: 59-73. Springer-Verlag.

M. Abadi A. Gordon. (1998). A calculus for cryptographic protocols: The Spi Calculus. Research Report 149, Digital Systems Research Center, Palo Alto, CA, USA.

M. Abadi and A. Gordon. (1998). A bisimulation method for cryptographic protocols. Nordic Journal of Computing, 5(4):267–303.

M. Abadi, A. D. Gordon. (1999). A calculus for cryptographic protocols: The spi-calculus. In Fourth ACM Conference on Computer and Communication Security, pages 36-47, ACM Press, 1999.

M. Baudet. (2005). Deciding security of protocols against off-line guessing attacks. In Proc. 12th Conference on Computer and Communications Security (CCS'05). ACM Press, 2005.

M. Borcale, R. De Nicola. and Rosario Pugiiese. (2000). Process algebraic analysis of cryptographic protocols lFlP Conference Proceedings Vol. 183: 375-392.

M. Boreale. Symbolic analysis of cryptographic protocols in the spi calculus. in: Proceedings of LICS '99, IEEE (1999),

R. Amadio, D. Lugiez. (2000). On the reachability problem in cryptographic protocols, In Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'00), 2000.

R. Focardi, M. Maffei (2004). The ρ-spi Calculus at Work: Authentication Case Studies. Work partially supported by MIUR project 'Modelli formali per la sicurezza' and EU Contract IST-2001-32617 'Models and Types for Security in Mobile Distributed Systems' (MyThS).

R. Milner. (1980). A Calculus of Communicating Systems, volume 92 of Lecture Notes in Computer Science. Springer-Verlag, New York, NY, USA, 1980.

R. Milner. (1989) Communication and Concurrency. Prentice Hall.

R. Milner, J. Parrow, and D. Walker. (1992). A calculus of mobile processes, parts I and II. Information and Computation, 100(1):1–40, 41–77, September 1992.

R. Milner. Communicating and Mobile Systems: The π-Calculus. Cambridge University Press, June 1999.

S. Delaune, S. Kremer, and M. D. Ryan. (2007). Symbolic bisimulation for the applied pi-calculus. In Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07), pages 133–145.

S. Matsuo, K. Miyazaki, A. Otsuka, and D. Basin. (2009) How to Evaluate the Security of Real-life Cryptographic Protocols? The cases of ISO/IEC 29128 and CRYPTREC.

S. Schneider. (1997) Verifying authentication protocols with CSP, In Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97), Rockport, Massachusetts, USA. IEEE Computer Society Press.

T. Kahsai. (2006). Towards semi automated equivalence checking of spi calculus processes. Master Thesis.

Y. Gu, Y. Fu and Guoqiang Li. (2005). A Simple Process Calculus for the analysis of Security Protocols. Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05) 0-7695-2405-2/05 © 2005 IEEE

X. Li; G. Zheng; L. Li; J. Wu; W. Chen. (2009). Stochastic extension for real time process algebra with urgency executing policy Proceedings - 2009 International Conference on Computational Intelligence and Software Engineering, CiSE 2009, pp.1 - 4.

Appendix

Appendix A.1

Details of Process Algebra and Operational Semantic

A.1.1 *Process Algebra*

Process algebras are executable specification languages for the description of concurrent systems, or processes. Their behavior is represented by a set of atomic *Input ,Output and Silent* actions that they can perform, each independently and in interaction with each other. A specification composes these actions via a few basic operators, like sequentialization (or prefix), parallel composition, and nondeterministic choice usually allowing communications. Furthermore there are some scope operators, such as restrictions and hiding. Process algebras are, usually, interpreted on an interleaving model, making concurrency not directly observable.

In computer science, the process calculi (or process algebras) are a diverse family of related approaches for formally modeling concurrent systems. Process calculi provide a tool for the high-level description of interactions, communications, and synchronizations between a collection of independent agents or processes. They also provide algebraic laws that allow process descriptions to be manipulated and analyzed, and permit formal reasoning about equivalences between processes (e.g., using bisimulation).

Process algebras have been often given an operational semantics in SOS (Structural Operational Semantics) style. The behavior and evolution of a program (process) are described through a ***transition system***. So, computations are formally defined as paths in the transition system. (for more details, refers to Appendix A).

Classical process algebras (among the others CCS (Milner 1989), CSP (Brookes et al. 1984), ACP (Bergstra and Klop 1984), Meije (Austry and Boudol

1984)) have been successively extended to cope with the possibility to exchange names (name passing) and processes (Higher Order calculi) in communications. These features express mobility, i.e. the dynamic change of the control structure of processes, and are present in the π-calculus (Milner et al. 1992), *HO*-π (Saniorgi 1992), the join calculus (Fournet et al. 1996), the ambient calculus (Nielson et al. 1999), the spi-calculus (Abadi and Gordan 1998) and many others.

A protocol is described as a process in process algebra, such as CSP and CCS, or, more recently in the π-calculus (Milner 1999; Milner et al. 1992) and the spi-calculus (Abadi and Gordan 1998). The desired security property is then studied by checking its specification on all its computations.

Remarkably, the semantics of process algebras is often given in a logical style, by defining a transition system (akin to a graph with labeled arcs, whose nodes represent states). So computations are formally defined as paths in the transition system. Moreover, the transition system associated with a protocol has often a finite number of states, in which case the analysis is mechanizable and provides complete answers. Considerable research has been done in recent years using various process algebras (without mobility) and equivalences, e.g., to establish properties about the information flow and to detect flows in protocols (Focardi and Gorrieri 1997). More recent work extends the above to study the foundations of secure mobile code; e.g. the works on the spi-calculus (Abadi and Gordan 1998).

Process algebras offer a pure framework to study concurrent and distributed systems and, in turn, the security issues connected to them. Systems are specified as expressions of the calculus, called *processes.* Processes are obtained by combining via a few operators (sequential and parallel composition, nondeterministic choice, declarations) the basic actions of sending and of receiving messages between processes along channels. Furthermore, there are some scope operators, such as restrictions and hiding.

By focusing on the essence of interactions among processes, these calculi make their well-established theory available to inquiry the subtle aspects of communications, in which security problems or flaws often hide. In this aspect, process algebras furnish a common background, where comparing different formulations and models for the same property is possible.

Among process algebras, the π-calculus (Milner 1999; Milner et al. 1992), a foundational calculus based on the notion of name passing, seems particularly suitable to address security problems, especially to model the usage of secret information. In its setting, names - that represent values, or messages, and also channels - can be created and passed. Processes can only communicate on the channels they know: for them, learning the name of a channel amounts to possessing the capability to communicate on the channel. A process can extend its communication possibilities, via an explicit mechanism of the calculus, called *scope extrusion*, which enlarges the scope of names. Consequently, the semantic rules may explicitly control the access to channels and to data.

Nevertheless, the π-calculus is yet too abstract to explicitly represent cryptographic operations, which are an important ingredient of security protocols. To fill this gap, Abadi and Gordon defined the spi-calculus (Abadi and Gordan 1998), an extension of the π-calculus with primitives for encryption and decryption.

The π-calculus is a model of computation for concurrent systems. The syntax of π-calculus lets you represent processes, parallel composition of processes, synchronous communication between processes through channels, creation of fresh channels, replication of processes, and non-determinism.

A.1.2 *Operational Semantics*

The operational semantics gives one of the major approaches, to the formal semantics. Introduced in the sixties, (McCarthy 1963; Lucas 1971), it describes the meaning of a programming language in terms of elementary steps on an abstract machine, formalized as transitions between states. Behaviors are graphically represented as transition systems, i.e. oriented graphs, where nodes represent states (or configurations) and arcs represent transitions between states. Transitions may be labeled by additional information on the activity performed.

Operational semantics provides quite a natural and intuitive way to understand how a program behaves. This aspect, joint with the simplicity of its mathematical basis, makes operational semantics helpful both in the design and implementation phases.

The main drawback is instead that the meaning of programs is not directly modeled, because it necessarily passes through their execution sequences. Therefore it is more difficult to reason about programs themselves, without concern of implementation and execution problems. This may lead to a lack of structure and of compositionality, i.e. the possibility to give the semantics of a construct in terms of the semantics of its components. Compositionality, essential for making the semantic definitions finite, appears in this framework only in the seventies, (De Bakker 1972), leading Plotkin (1981) to propose a structural approach.

A.1.2.1 *Structural Approach*

The introduction of Structural Operational Semantics or SOS (Plotkin 1981) arises from the cross-fertilization with denotational techniques, borrowing from them notions such as compositionality and abstract syntax. The semantics of compound constructs is defined in terms of the semantics of their components, following the various syntactic possibilities, i.e. it is syntax-directed. Transitions are deduced in a logically - based way, by inducing on the syntactic constructs. Transitions themselves are defined using axioms and inference rules of the form

$$\frac{\text{Premises}}{\text{Conclusion}}$$

where, if the premises are satisfied, so does the conclusion. From a formal point of view, this amounts to having a logically - based proof system, that, in particular, provides a reliable support to the development of automatic tools. (for more details, refers to Appendix A)

The structural operational semantics, still preserving the simplicity of the traditional operational semantics, exploits compositionality, making specifications modular and modules suitable for re-use.

Finally, it is sufficiently general to allow tuning the level of abstraction, without drastically changing the method.

A.1.2.2 *Enhanced Operational Semantic*

Descriptions of systems need to be more detailed and concrete in the implementation phase. They require considering also information on the external environment, concerning architecture and topology. Aspects, such as locality or causality, become of interest, from this point of view. In particular, locality-based semantics model distributed systems according to their physical or geographical distribution, localizing each activity in the site it is performed.

To capture this kind of low-level information, one often resorts to true concurrent models, because the interleaving ones only capture more extensional aspects. Unfortunately, the non-interleaving approach presents a more difficult formal treatment (Reisig 1985). This little excursus shows that both approaches are essential to understand distributed systems, for complementary reasons.

Different strategies have been used to integrate the two philosophies (e.g. (Degano & Montanari 1987)). Some extensions to transition systems towards true concurrency have been presented, among which that of *Proved transition systems* (Degano & Priami 1992, 1999, 1996) (See Section 3.2). A proved transition is labeled by an encoding of its deduction tree or proof term. Proved transition systems can be considered as a sort of logogrammatic representation of computations, containing all the possible encodable information. From this concrete and unifying model, interleaving in style, it is possible to retrieve a large number of different semantic models by abstracting from undesired information (Bodei et al. 1998; Bodei & Priami 1997; Degano & Priami 1999). Aspects like causality or locality may be retrieved in this way. This approach has the advantage that the theory, the techniques and the tools valid for the interleaving models are still applicable.

A.1.3 *The Reaction Relation*

Based on (Abadi and Gordan 1998), we define the *reaction relation* in three phases. In the first one, we have the definition of the *reduction relation*, >. In the second one, we define what is called *structural equivalence* of two processes, ≡, whereas in the third one, we present the definition of the *reaction relation*, →.

Definition A.1.3.1 The *reduction relation*, $> \subseteq Proc \times Proc$, is defined as the least relation on closed processes defined by the following rules:

(RedRepl) $!P \; > \; P \mid !P$

(RedMatch) $[M \text{ is } M] .P \; > \; P$

(RedPair) $let\ (x, y) = (M, N)\ in\ P \; > \; P[M/x][N/y]$

(RedZero) $\textit{case } 0 \textit{ of } 0: x\ \textit{suc}(P): Q > P$

(RedSuc) $\textit{case suc}(n) \textit{ of } 0: x\ \textit{suc}(P): Q > Q[n/x]$

(RedDecrypt) $\textit{case } \{t\}k \textit{ of } \{x\}k \textit{ in } P > P[t/x]$

Informally, we say that two processes are *structural equivalent* if one can be transformed into the other using the rules below.

Definition A.1.3.2 The *structural equivalence,* $\equiv \subseteq Proc \times Proc$, is defined as the least relation on closed processes that satisfies the following equations and rules:

(StructNil) $P \mid \mathbf{0} \equiv P$

(StructComm) $P \mid Q \equiv Q \mid P$

(StructAssoc) $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$

(StructSwitch) $(\nu m)(\nu n)\, P \equiv (\nu n)(\nu m)\, P$, if $n \neq m$

(StructDrop) $(\nu m)\, \mathbf{0} \equiv \mathbf{0}$

(StructExtrusion) $(\nu m)(P \mid Q) \equiv P \mid (\nu m)Q$, if $m \notin fn(P)$

$$\frac{P > Q}{P \equiv Q}(StructRed) \qquad \frac{}{P \equiv P}(StructRefl)$$

$$\frac{P \equiv Q}{Q \equiv P}(StructSymm) \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R}(StructTrans)$$

$$\frac{P \equiv P'}{P \mid Q \equiv P' \mid Q}(StructPar) \qquad \frac{P \equiv P'}{(\nu m)P \mid \equiv (\nu m)P'}(StructRes)$$

We are now ready to define the *reaction relation* of two closed processes. The previous relations were defined to allow the rearrangement of processes so that the reaction could be possible.

Definition A.1.3.3 The *reaction relation* on closed processes, $\rightarrow \subseteq Proc \times Proc$, is defined as the least relation on closed processes that satisfies the following axiom,

$\bar{c}\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q[M/x]$ (ReactInter), and the following rules:

$$\frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \equiv Q}(ReactStruct)$$

$$\frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}(ReactPar) \qquad \frac{P \rightarrow P'}{(\nu n)P \rightarrow (\nu n)P'}(ReactRes)$$

A.1.4 *Commitment Relation in the Spi-calculus*

In order to define the *commitment relation,* we need two new syntactic forms - *abstractions* and *concretions*. An *abstraction* is an expression of the form (*x*).*P* where *x* is a bound variable and *P* is a process. When *F* is the abstraction (*x*).*P* and *M* is a term, we write *F*(*M*) for *P*[*M/x*].

A *concretion* is an expression of the form $(\nu \vec{n})\langle M \rangle P$ where *M* is a term; *P* is a process and $\vec{n}$ are names that are bound in *M* and *P*. We will use *C* and *D* for concretions.

We define an *agent* as an abstraction, a concretion or a process. We will use the variables *A* and *B* when representing agents, and define *fv*(*A*) and *fn*(*A*) as the sets of free variables and free names of an agent *A*, respectively. The definitions of *fv*(*A*) and *fn*(*A*) are the expected extensions of the Notation 1, Definitions 3.2.1.1 and 3.2.1.3.

We now have to extend restriction and composition to arbitrary agents. We will do this using the following rules:

$$(\nu\, m)(x).P ::= (x)(\nu\, m)P;$$

$$R \mid (x).P \quad ::= (x).(R \mid P) \qquad if \;\; x \notin fv(R)$$

$$(\nu\, m)\,(\nu\, \vec{n})\langle M \rangle P \;::= \begin{cases} (\nu\, m, \vec{n})\langle M \rangle P & if \;\; m \in fn(M) \\ (\nu\, \vec{n})\langle M \rangle (\nu\, m)\; P & otherwise \end{cases}$$

$$R \mid (\nu\, \vec{n})\langle M \rangle P ::= (\nu\, \vec{n})\,\langle M \rangle (R \mid P) \qquad if \;\; \{\vec{n}\} \cap fn(R) = \emptyset$$

In the first and third equations, we also suppose that $m \notin \{\vec{n}\}$. We define the dual composition *A* | *R* symmetrically.

The *interactions* of an abstraction *F* = (*x*).*P* and a concretion *C* = $(^{\nu\, \vec{n}}\,)^{\langle M \rangle P}$, *F@C* and *C@F*, are defined as the processes:

$$F@C ::= (\nu\, \vec{n})(P[M/x] \mid Q);$$

$$C@F ::= (\nu\, \vec{n})(Q \mid P[M/x]).$$

Intuitively, these processes represent the interaction of *P* and *Q*. It is the same as *P* and *Q* running in parallel and communicating using the same channel *c*.

We will now define how transitions work in Spi-calculus.

Definition A.1.4.1 We say that β is a *barb* if it is a name *m* (representing an input) or a co-name $\overline{m}$ (representing an output). We say that β is an *action* if it is a barb or the *silent action* τ.

Now we are able to introduce the *commitment relation* as in (Abadi and Gordan 1998).

Definition A.1.4.2 The *commitment relation*, $\rightarrow$, is written $P \xrightarrow{\alpha} A$, where *P* is a closed process, α is an action and *A* is a closed agent, and is defined inductively by the following rules:

$$\frac{}{m(x).P \xrightarrow{m} (x).P}\,(\text{CommIn})$$

$$\frac{}{\overline{m}\langle M\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M\rangle.P}\,(\text{CommOut})$$

$$\frac{P \xrightarrow{m} F \qquad Q \xrightarrow{\overline{m}} C}{P \mid Q \xrightarrow{\tau} F@C}\,(CommInter1)$$

$$\frac{P \xrightarrow{\overline{m}} C \qquad Q \xrightarrow{m} F}{P \mid Q \xrightarrow{\tau} C@F}\,(CommInter2)$$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q}\,(CommLPar)$$

$$\frac{P > Q \qquad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}\,(CommLRed)$$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} P \mid A}\ (CommRPar)$$

$$\frac{P \xrightarrow{\alpha} A}{P \mid Q \xrightarrow{\alpha} A \mid Q}\ (CommLPar)$$

$$\frac{P \xrightarrow{\alpha} A \qquad \alpha \notin \{m, \overline{m}\}}{(\nu m)P \xrightarrow{\alpha} (\nu m)A}\ (CommL\,Res)$$

Whenever $P \xrightarrow{\alpha} A$, and the action α is a name, then *A* is an abstraction, when α is a co-name, *A* is a concretion, and when α is τ, *A* is a process. Therefore the commitment relation indexed by τ is a binary relation on *Proc*. We write $P \xrightarrow{\tau} \equiv Q$ when there exists a process *R* such that $P \xrightarrow{\tau} R$ and $R \equiv Q$.

Proposition A.1.4.1 $P \to Q$ *if and only iff* $P \xrightarrow{\tau} \equiv Q$.

Proof:

For the backwards direction suppose $P \xrightarrow{\tau} R$ and $R \equiv Q$, then $P \to R$ and then $P \to Q$ by (React Struct). We can show that $P \to Q$ implies that there exists *R* such that $P \xrightarrow{\tau} R$ and $R \equiv Q$ by induction on the derivation of $P \to Q$. The only interesting case is (React Struct). Suppose that $P \to Q$ follows from $P \equiv P'$, $P' \to Q'$ *and* $Q' \equiv Q$. By induction hypothesis, $P' \xrightarrow{\tau} Q''$ with $Q'' \equiv Q'$.

By **Definition A.1.4.2**, structural equivalence is a strong bisimulation, so $P \xrightarrow{\tau} R$ for *R* such that $R \equiv Q''$. This with the previous equation gives $R \equiv Q$ as required.