Preface

This M.Sc. Thesis presents the results of my final master project entitled *Static Analysis for Protocol Validation in Hierarchical Networks*. The master project was carried out in the period of 18th of January to 27th of July 2005 and corresponds to 30 ETCS points. It has been the final part of my studies for the Master of Science degree at the Department of Informatics and Mathematics Modelling (IMM), Technical University of Denmark (DTU).

I would like to take this opportunity to thank my supervisor, Professor Hanne Riis Nielson, for her counseling, support and not least her interest throughout the period of my M.Sc. Thesis project.

I would also like to thank Flemming Nielson, Henrik Pilegaard, Terkel Tolstrup, Esben Heltoft Andersen and Christoffer Rosenkilde Nielson for comments and discussions during my thesis work.

Last but not least, I would like to thank my wife, Shanshan Song, for her proof reading as well as emotional support.

Lyngby, July 27th, 2005

Ye Zhang

Abstract

The ambient calculus presents a high-level view of mobile computation and gives rise to a high-level treatment of the related security issues. Ambients, which are named bounded places, allow us to model the concept of local networks in a natural way. The tree structure formed by ambients just captures the characteristic of our interested network, the hierarchical network.

In this M.Sc. thesis, we focus on validating cryptographic protocols working on hierarchical networks. We begin with extending Boxed Ambients with annotations which declare the authentication intentions of cryptographic protocols. We then develop a static analysis for tracking the interested properties of a process. The analysis specification is defined in Flow Logic in order to separate the definition of specification from the actual implementation. Subject to the environment of the hierarchical network, we declare the capability of the attacker of the hierarchical network based on the Dolev-Yao attacker.

The program analysis is fully implemented to compute least estimates for a process. Here we use the Succinct Solver as our constraint solver. It computes the least interpretation of predicate symbols given formulae written in Alternative-free Least Fixed Point logic (ALFP). With this tool, we validate symmetric key protocols applied on hierarchical networks.

Keywords: Boxed Ambients, Static Analysis, Hardest Attacker, Flow Logic, Hierarchical Network, Cryptographic Protocol.

Table of Contents

1 Introduction

Nowadays millions of local networks of companies, colleges, etc. are connected through the Internet. This actually gives the whole network a hierarchical structure, a main characteristic of networks. A typical network is illustrated in Figure 1-1. As illustrated, the Internet connects internet users directly as well as all kinds of local networks, such as the home network and the office network which can further have sub-networks.

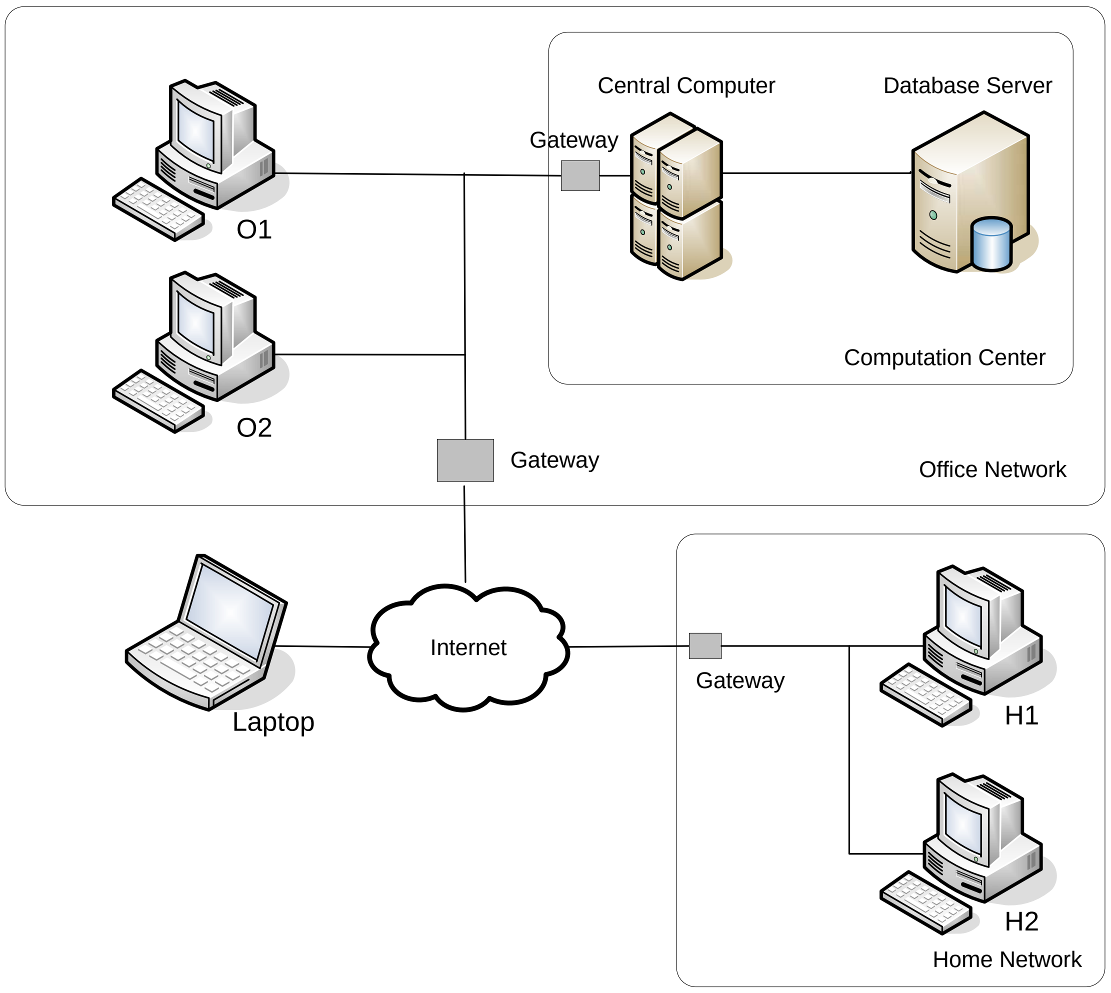


Figure 1-1: **An example of a network with hierarchical structure**

By inserting gateways in between two networks the figure also illustrates another important property of the network: Packets transferred between computers will never be leaked to devices outside if the sources and destinations of these packets are located in the same local network. Intuitively this property can be visualized as the boundary of the network - the rectangles in the figure, by which all devices and sub-networks of the network are enclosed.

In fact, to present the network hierarchy the sub-network boundary should strictly be enclosed by the parent network boundary. In next chapter, we shall see how the above properties and concepts are formalized by ambient calculus perfectly.

The hierarchical structure of the network gives rise to the complexity of protocol validation because participants of a protocol may show up on the different locations and communication varies from one local network to another local network.

This M.Sc. Thesis investigates and proposes a formal and automatic way to validate cryptographic protocols applied in hierarchical networks. Basing on the analysis results, we can find the flaw of a protocol and then fix the security issues. Furthermore we can optimize a protocol to improve its performance while still ensuring our modification does not introduce any flaw by validating the optimized protocol automatically.

1.1 Motivation and Objectives

Many formal analyses of cryptographic protocols, such as LySa [2], on-the-fly model checker (OFMC) [35] and strand spaces [36], etc., assume that all participants show on the same public-accessed network. This flat space of locations of the analyzed network limits the use of these techniques when the idea of local networks is introduced. The first motivation of this thesis is then to declare a calculus which is able to model protocols applied in hierarchical networks so that a static analysis can pinpoint many kinds of flaws in cryptographic protocols. There are four common properties considered when validating a cryptographic protocol:

(1) **Confidentiality**. A protocol guarantees confidentiality if only authorized participants can read protected data.

(2) **Authenticity**. Authenticity requires that sensible information is sent and received by the participant intended by the protocol.

(3) **Integrity**. In the context of cryptographic protocols integrity means unmodified. That is any change to the sensible message can be detected by the intended participant.

(4) **Availability**. If any participant can always respond to others' request timely, then it ensures availability.

In this thesis we are interested in authentication property and take the assumption of perfect cryptography.

To validate a cryptographic protocol, we need properly defined attackers to conduct all kinds of attacks to the protocol. As we have pointed out communication varies from one network to another in hierarchical networks. But the classical Dolev-Yao conditions [7] is designed only for the environment of public accessed network. This leads to an adjustment to the Dolev-Yao condition.

We shall also consider private key storage and retrieving as part of our protocol validation. Most formal analyses only validate protocol communication. However key storage and retrieve could be the weak point when a protocol is adopted in a real network. Thus we shall model them and analyze whether or not any keys could be acquired by unauthorized participants or attackers.

1.2 Technical Context of Formalism

Our calculus can be considered as a variant of the ambient calculus. Many existing calculi have inspired our work. In particular:

- Mobile Ambients [5, 6]. First introducing the participant of ambients, it presents a high-level abstract of mobile computing and computation. The design of mobile ambients focuses on four basic notions: location, mobility, authorization to move and communication via the shared location. Among them, a topology of boundaries presenting the existence of separate locations (formalized as ambients in the calculus) is very expressive to model the structure of hierarchical networks.
- Boxed Ambients [9,8]. As a variant of Mobile Ambients Boxed Ambients allows communication between adjacent layers of ambients. Accordingly an ambient can read message not only from its local communication box but also from its parent's or children's communication box. In particular to read message from the communication box of its children, an ambient has to know the name of its children. This nice feature gives rise to a flavor of decryption. Based on Boxed Ambients, the research in [3] first presents how ambient primitives can be used for cryptographic purpose. It showed symmetric cryptography and many key exchange protocols can be expressed in Boxed Ambients. Our calculus aiming at validating cryptographic protocols will be based on Boxed Ambients.
- LySa-calculus [2]. The work in [2] begins with extending standard protocol narrations with annotations which declare the authentication intensions explicitly. The annotated protocol narrations are then translated into terms of the process algebra LySa so that the protocol can be validated by well-defined static analysis. Our approach acquires inspiration from LySa-calculus in four main aspects:

  (1) Syntax. Following LySa, we have annotations declaring authentication intensions.

  (2) Semantics. Inspired by LySa, we define reference monitor semantics as well as standard semantics for ABoxed Ambients. Reference monitor semantics defines in which case the executing control flow analysis should be aborted.

  (3) Modeling of the attacker. The approach that LySa uses to model an attacker has provided inspiration for us to formalize our attacker.

  (4) Implementation. Since both LySa-tool and our tool adopt the Succinct Solver to calculate the sets of approximations, the implementation of LySa

also functions as a good example for us to implement our analysis in Standard Meta Language (SML).

1.3 Report Roadmap

This thesis consists of seven chapters of which this introduction is the first.

A brief presentation of the remaining six chapters is given as the followings:

Chapter 2 introduces the concepts of ambients, the syntax and semantics of ABoxed Ambients. We also discuss how to use ambient calculus for cryptographic purpose. Last we choose Wide Mouthed Frog (WMF) [4] to demonstrate the use of ABoxed Ambients for formalizing cryptographic protocols.

Chapter 3 develops a static analysis for tracking the set of encrypted packets which are successfully being decrypted at each relevant ambient. The specification of the static analysis is declared in Flow Logic. Moreover we show the semantic correctness of the analysis.

Chapter 4 adjusts the Dolev-Yao conditions [7] to model the ability of attackers in the context of hierarchical networks. We then show the approach realizes a hardest attacker [11,10].

Chapter 5 sketches our implementation for constraint based program analysis [1]. We present the steps of defining the generation function which given an ABoxed process returns a formula in Alternation-free Least Fixed Point logic (ALFP). We then present the Indexed ABoxed Ambients which helps us to declare processes modeling man-in-the-middle attack.

Chapter 6 presents the experiment results acquired by using our analysis to validate many symmetric key protocols. We compare our analysis estimates with those of LySa. Furthermore we illustrate how to use the analysis to assist the design of the protocol in a hierarchical network.

Chapter 7 presents the conclusion and contribution of the project. Future work is also discussed in this chapter.

2 ABoxed Ambients

At the beginning of this chapter, we analyze the characteristics of the structure of hierarchical networks. We then give the definition of ambients and show the relation between the structure of hierarchical networks and that formed by ambients.

Next we present our process calculus, ABoxed Ambients, by giving its formal syntax and semantics. For the purpose of validating protocols, we syntactically divide all ambients into two classes: ambients for agents and local networks (they are also called sites in this thesis) and ambients for packets transferred between participants. We shall use these definitions to develop our static analysis and define our attacker in the next two chapters.

One problem which keeps us from using ambient calculi for cryptographic purpose is that they are scare of primitives for decryption and encryption. To verify authenticity of a protocol we however need some clearly defined processes for expressing decryption and encryption. Accordingly we could further determine where to add annotations since they are only needed for the encryptions or decryptions. In this chapter we investigate the possible ways to formalize decryption and encryption as ambient calculus. We end our discussion of this chapter by presenting an ABoxed Ambients specification of Wide Mouthed Frog (WMF) protocol.

2.1 Principles: Hierarchical Networks *vs.* Ambients

We have used the word "***Hierarchical***" several times in Chapter 1 to emphasize the special structure of the networks and in Figure 1-1 we give an example of such networks. But what is the essential property of hierarchy? To answer this question, let us consider the example network in Figure 1-1 again. If we treat all participants of a protocol, local networks and the whole network (including all stuff on the network) as a set of nodes, then we could build up a tree structure as below.

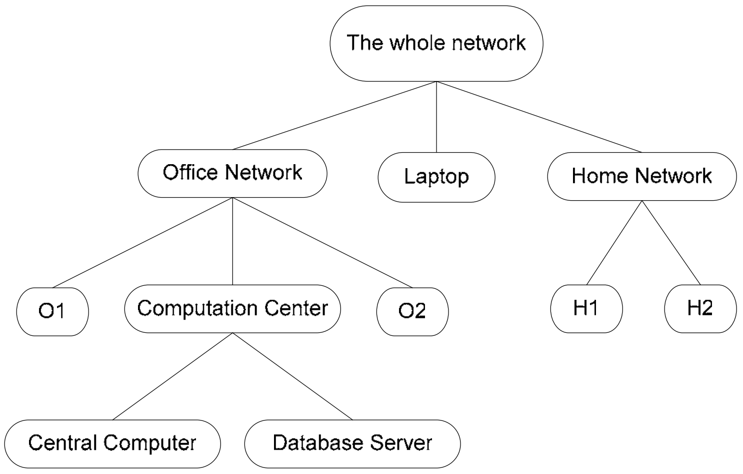


Figure 2-1: Tree structure of the network shown in Figure 1-1

The branches can be understood as "enclosing" relation between the upper node and the lower one. For example, the whole network encloses the office network which then encloses the computer O1, and so on.

We use the above example to show that the idea behind hierarchy is the *tree structure*. Actually hierarchical structure is very popular in computer systems. For example, file systems, where directories have files and subdirectories beneath them.

An ***ambient*** is a named bounded place where computation happens. The boundary of an ambient determines what is outside and what is inside. For example we could consider a laptop (bounded by its case and ports) as an ambient.

Formally ambients have following structure:

- Each ambient contains a collection of threads (also known as processes) running in parallel. These threads, as a whole, move with their enclosing ambient;
- Each ambient has a name by which other ambients can access to the ambient. The names of ambients are unforgeable;
- Each ambient contains a number of subambients. Each subambient has its own name, threads and subambients.

From the view of the structure of ambients, the most interesting characteristic of ambients is that they can be nested inside another ambient and form a tree structure. Thus ambients are able to model hierarchical networks with same or similar hierarchical structure. In other words, we can capture the characteristic of hierarchical network with ambients. Now the network in Figure 1-1 can be formalized with ambients as:

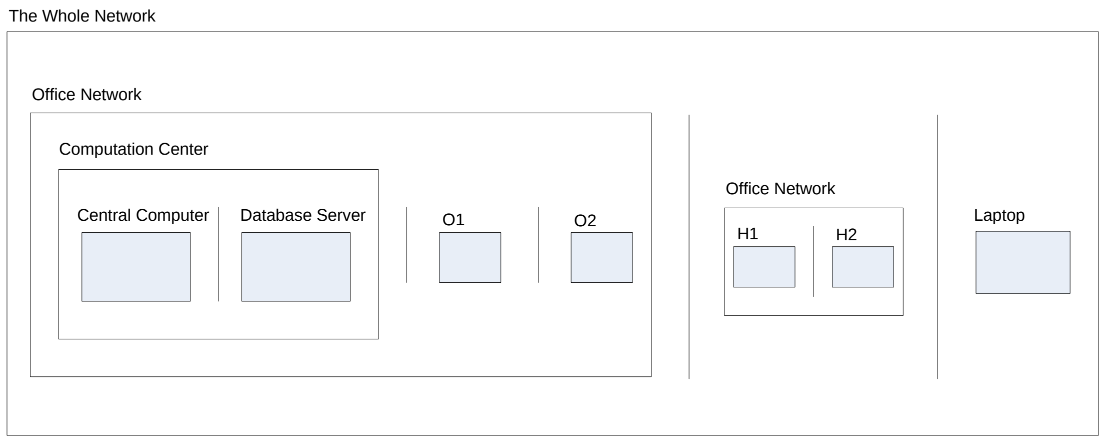


Figure 2-2: Ambient structure of the network in Figure 1-1.

Now "enclosing" relation is visualized as such that the boundary of an ambient directly surrounds that of another one. As the figure presents, ambients also construct a tree structure which completely matches the tree structure in Figure 2-1. Intuitively the ambient for the whole network provides a *background environment* within which all computation is supposed to happen but no ambients enclosed directly or indirectly by the *background environment* can move out of it. We shall give formal definition for the background environment after presenting the syntax and semantics of our calculus.

In the context of protocol validation specifically, we have two main classes of ambients:

(1) Site. For instance, laptops, desktops, servers and databases are typical sites. Local networks are also thought as sites which are bounded by all stuff connected by the corresponding local network. Intuitively some of sites may move around, i.e. laptops. Others, in contrast, seldom change their locations, i.e. local networks, desktops, servers, etc. Here we shall not allow any site to move in order to avoid handling too complex cases in defining attackers. In particular attackers are considered as sites; it means they can not change their locations by themselves.

(2) Packet. In cryptographic protocol, a packet is a single data object (bounded by 'self') moving from one site to another. A packet may output messages to its enclosing site but it has no primitives of input. This assumption works for modeling of cryptographic protocols since no executable code is enclosed by packets and thus a packet is as simple as a movable data.

2.2 Syntax

ABoxed Ambients consists of five components: processes P, capabilities M, namings N, and communication direction η. The syntax of processes P is as below

$P ::= (\nu\ C)P$	introduces a process with private name C
$\mid\ (\nu k\ n)P$	introduces a process with secret name n
	which is transparent to the attacker within the scope of C
$\mid\ 0$	the inactive process
$\mid\ P_1 \mid P_2$	parallel composition
$\mid\ !P$	replication: any number of occurrences of P
$\mid\ N[P]$	ambient containing P and named N $(N \in \mathcal{N})$
$\mid\ M.P$	a capability M followed by P
$\mid\ <M_1,\cdots,M_k>^\eta$	asynchronous polyadic output
$\mid\ (M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^\eta.P$	input with matching
$\mid\ \bullet$	attacker process

Table 2-1: **The syntax of ABoxed Ambients**

Here $(\nu\ C)P$ introduces a new private name C into process P. It is known by the attacker who is inside P. We also have an operation $(\nu k\ C)P$ which is same as $(\nu\ C)P$ except that to the attacker inside P secret name C is not counted as part of his initial knowledge any more. Its function will become clearer after we present our attackers in Chapter 4.

The naming N of an ambient name can be either a constant or a variable but not both and that is defined as:

$N ::= C$	name	$(C \in \mathcal{C})$
$\mid \; x$	variable	$(x \in \mathcal{X})$

where $N \in \mathcal{V}$ and $\mathcal{X} = \mathcal{V} \setminus \mathcal{C}$.

The distinction between constant and variables (introduced by input) adds clarity both to semantics and to the analysis. Furthermore among constants we distinguish between the constants for site names and those for non-site names as:

$C ::= n_s$	site name	$n_s \in \mathcal{C}_s$
$\mid \; n$	non-site name	$n \in \mathcal{C}_P$

where $\mathcal{C}_P = \mathcal{C} \setminus \mathcal{C}_S$

Following [9, 8] auxiliary syntactic category η is introduced for the communication direction as:

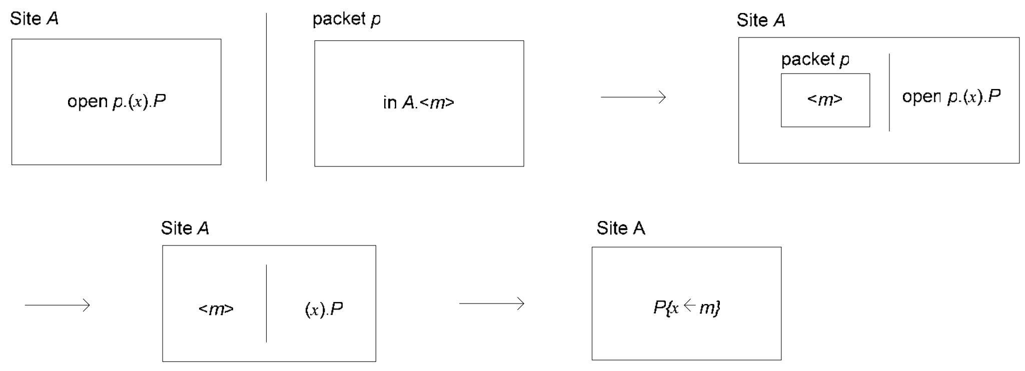
$\eta ::= N$	names and variables
$\mid \; \uparrow$	enclosing ambients
$\mid \; \circ$	local

Considering that the message composed in cryptographic protocols is often polyadic we adopt polyadic communication here. In particular inspired by Lysa calculus [2], we introduce a simple form of patterns, $(M'_1,\cdots,M'_j; x_{j+1},\cdots,x_k)^\eta$, to be matched against a k-tuple of values $(M_1,\cdots,M_k)$. If the first $1 \le i \le j$ values M_i pairwise correspond to the values M'_i , the matching should succeed and accordingly the remaining k - j values are bound to the variables $x_{j+1},\cdots,x_k$. The pair matching and binding are syntactically separated by a semi-colon. For example, let P = $<\mathrm{y}>^\circ \mid (x;)^\circ$ then the communication succeeds only if $y = x$. If we modify the process a little to be $<\mathrm{y}>^\circ \mid (;x)^\circ$, this time the communication always succeeds as no check is needed here and as the result variable x will bind with the value y.

The capabilities M are declared as:

$M ::=$ in N	enter siblings named N
$\mid$ out N	exit from parent ambient N
$\mid \; N$	naming

Here we have in-capability and out-capability but not open-capability following the designers of Boxed Ambients. Open-capability is required in Mobile Ambients since it only allows local communication: If two ambients need communicate with each other, they have to form a shared location at first. That is one of them must be 'opened' by the other. The whole progress could be graphically presented as



The first step shows that the packet must move into the site *A* before it can be opened by the site *A*. Then the site *A* opens the packets. Last the local communication is executed and *x* is bound with the message *m*.

In Boxed Ambients, however, communication primitives between adjacent layers are introduced and thus open-capability is, in that sense, not necessary. Furthermore to simplify our presentation our calculus is different from Boxed Ambients in excluding the formation of composite capabilities, such as nil capabilities ε and the capability concatenation $M_1.M_2$. Also since we are interested in authenticity of the security properties of cryptographic protocols, we do not adopt co-capabilities as would be needed for expressing access control in Safe Ambients [32], Discretionary Ambients [3], BioAmbients [33], etc.

We use the solid circle • to declare our assumption about which locations the attacker may access to. In LySa only one location is assumed for the attacker because the analyzed network has a flat structure. However we are working with hierarchical network now. Thus it is necessary to explicitly specify which location they can access to (besides the location for the Internet).

Different from the calculi in [3], we have not the concept of group. But we still need similar concept to make sure the result of our control flow analysis makes sense in $\alpha-$conversion systems. We thus borrow the idea from LySa and introduce canonical representative for each naming. Specifically we declare that there is a canonical name $\lfloor C \rfloor$ for each name *C*. Similarly each variable also has a corresponding canonical variable. We impose that two names or variables are $\alpha-$convertible if and only if they have the same canonical name or variable respectively. The operation $\lfloor \cdot \rfloor$ is extended in a pointwise way to sets of names or variables. We further extend it to capabilities: $\lfloor M \rfloor$ is the capability where the name or variable is replaced by its canonical version. If there is no ambiguity then we shall use *M* for $\lfloor M \rfloor$ to avoid overloading our notation.

To describe the intentions of protocols in ABoxed Ambients, we annotate processes with crypto-points (labels attached to program points where encryptions and decryptions occur) and, origin and destination of encrypted messages. In the syntax of our calculus, we shall only annotate local output and input (from child ambients) in ABoxed Ambients as:

$<M_1, ..., M_k >^{\circ}_{\ell}$ [dest $\mathcal{L}$]

$(M'_1, ..., M'_j; x_{j+1}, ..., x_k)^{n}_{\ell}$[orig $\mathcal{L}$]

where crypto-points ℓ are from some enumerable set $\mathcal{D}$, which is disjoint from $\mathcal{N}$ and $\mathcal{X}$ and includes a special symbol ε expressing a trivial crypto-point. The assertion [dest $\mathcal{L}$] specifies all intended crypto-points $\mathcal{L} \subseteq \mathcal{D}$ for the decryption of the encrypted value and [orig $\mathcal{L}$] presents the encryption points $\mathcal{L} \subseteq \mathcal{D}$ where a message is allowed to have been encrypted. Following the conventions in LySa [2] we shall simply write [dest ℓ] and [orig ℓ] instead of the standard [dest $\{\ell\}$] and [orig $\{\ell\}$] if no misunderstanding is introduced. This completes the definition of the syntax of ABoxed Ambients. It is reasonable to postulate that other communication primitives can be used to express encryption and decryption. As we need some knowledge about the semantics of the calculus before discussing other possibilities, we shall present the relevant discussion after presenting our semantics.

Example 2.1 Suppose there are two sites *A* and *B* and a packet *p* inside *A* will be sent from *A* to *B*. Then this could be formalized as ABoxed Ambients process as:

$$A[\,p[\text{out } A.\text{in } B]] \mid B[\,]$$

As the process presents, we use three ambients named as *A*, *B* and *p* to formalize site A, site B and the packet p respectively. The mobility of the packet is expressed by the capabilities of ambients *p*. Intuitively these capabilities direct the packet *p* to move out of site *A* and then enter site *B*. We shall illustrate the evolvement of the process formally after presenting the semantics of ABoxed Ambients.

□

2.3 Semantics

Following the standard approach of the π-calculus, the semantics of ABoxed Ambients is specified by a structural congruence between processes, $\equiv$, and a reduction relation $\rightarrow$.

The structural congruence, $\equiv$, is defined on processes to be the least congruence satisfying the conditions listed in Table 2-2 on next page.

$P \equiv P$	$!P \equiv P \mid !P$	
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	$!0 \equiv 0$	
$P \equiv Q \Rightarrow Q \equiv P$		
	$(\nu\ n)0 \equiv 0$	
$P \equiv Q \Rightarrow (\nu\ n)P \equiv (\nu\ n)Q$	$(\nu\ n)(\nu\ n')P \equiv (\nu\ n')(\nu\ n)P$	if $\mathrm{n} \neq \mathrm{n}'$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$(\nu\ n)(P \mid Q) \equiv P \mid (\nu\ n)Q$	if $n \notin \mathrm{fn}(P)$
$P \equiv Q \Rightarrow !P \equiv !Q$	$(\nu\ n)(N[P]) \equiv N[(\nu\ n)P]$	if $n \notin \mathrm{fn}(N)$
$P \equiv Q \Rightarrow N[P] \equiv N[Q]$	$(\nu\ n)P \equiv (\nu\ \mathrm{m})(P\{n \leftarrow m\})$	if $\mathrm{m} \notin \mathrm{fn}(P)$
$P \equiv Q \Rightarrow M.P \equiv M.Q$		
$P \mid Q \equiv Q \mid P$		
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$		
$P \mid 0 \equiv P$		
$P \equiv Q \Rightarrow (x_1, \cdots, x_k)^\eta.P \equiv (x_1, \cdots, x_k)^\eta.Q$		

Table 2-2: Structural Congruence; $P \equiv P'$

The structural congruence relation allows rearranging the syntactic appearance of processes and performing α-renaming over private names. Following the idea of [3] we here restrict α-renaming only to private names.

The function fn(*P*), which collects all the free names in a process *P*, is defined in Table 2-3.

$\mathrm{fn}((\nu\ n\ \mu)P) \triangleq \mathrm{fn}(P) - \{n\}$	$\mathrm{fn}((\nu k\ \mu)P) \triangleq \mathrm{fn}(P) - \{n\}$
$\mathrm{fn}(0) \triangleq \varnothing$	$\mathrm{fn}(\mathrm{in}\ N) \triangleq \mathrm{fn}(N)$
$\mathrm{fn}(P_1 \mid P_2) \triangleq \mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)$	$\mathrm{fn}(\mathrm{out}\ N) \triangleq \mathrm{fn}(N)$
$\mathrm{fn}(!P) \triangleq \mathrm{fn}(P)$	$\mathrm{fn}(n) \triangleq \{n\}$
$\mathrm{fn}(N[P]) \triangleq \mathrm{fn}(N) \cup \mathrm{fn}(P)$	$\mathrm{fn}(x) \triangleq \varnothing$
$\mathrm{fn}(M.P) \triangleq \mathrm{fn}(M) \cup \mathrm{fn}(P)$	
$\mathrm{fn}(<M_1, \cdots, M_\mathrm{k}>^\eta) \triangleq \mathrm{fn}(M_1) \cup \cdots \cup \mathrm{fn}(M_\mathrm{k})$	
$\mathrm{fn}((M_1, \cdots, M_\mathrm{j}; x_{\mathrm{j}+1}, \cdots, x_\mathrm{k})^\eta.P) \triangleq \mathrm{fn}(M_1) \cup \cdots \cup \mathrm{fn}(M_\mathrm{j}) \cup \mathrm{fn}(P)$	

Table 2-3: **Free names; fn(*P*).**

As the variables are introduced by inputs, the function fv(*P*) is defined in Table 2-4 to collect all free variables in a process *P*.

$\mathrm{fv}((\nu\ n)P) \triangleq \mathrm{fv}(P)$	$\mathrm{fv}((\nu k\ n)P) \triangleq \mathrm{fv}(P)$
$\mathrm{fv}(0) \triangleq \varnothing$	$\mathrm{fv}(\mathrm{in}\ N) \triangleq \mathrm{fv}(N)$
$\mathrm{fv}(P_1 \mid P_2) \triangleq \mathrm{fv}(P_1) \cup \mathrm{fv}(P_2)$	$\mathrm{fv}(\mathrm{out}\ N) \triangleq \mathrm{fv}(N)$
$\mathrm{fv}(!P) \triangleq \mathrm{fv}(P)$	$\mathrm{fv}(n) \triangleq \varnothing$
$\mathrm{fv}(N[P]) \triangleq \mathrm{fv}(N) \cup \mathrm{fv}(P)$	$\mathrm{fv}(x) \triangleq \{x\}$
$\mathrm{fv}(M.P) \triangleq \mathrm{fv}(M) \cup \mathrm{fv}(P)$	
$\mathrm{fv}(<M_1, \cdots, M_k>^{\eta}) \triangleq \mathrm{fv}(M_1) \cup \cdots \cup \mathrm{fv}(M_k)$	
$\mathrm{fv}((M_1, \cdots, M_j; x_{j+1}, \cdots, x_k)^{\eta}.P) \triangleq \mathrm{fv}(P) - (\mathrm{fv}(x_{j+1}) \cup \cdots \cup \mathrm{fv}(x_k))$	

Table 2-4: **Free variables; fv(*P*).**

Both definitions of fn(P) and fv(P) are straightforward but presented for the completeness. Especially the attacker process is omitted or for now one can simply consider it as the inactive process 0 in the calculation. We shall discuss the free names and variables of the attacker in detail in Chapter 4.

The transition relation is summarized as two tables, Table 2-5 and Table 2-6.

(New)
$$\frac{P \to_{\mathcal{R}} Q}{(\nu\ n)P \to_{\mathcal{R}} (\nu\ n)Q}$$

(News)
$$\frac{P \to_{\mathcal{R}} Q}{(\nu k\ n)P \to_{\mathcal{R}} (\nu k\ n)Q}$$

(Amb)
$$\frac{P \to_{\mathcal{R}} Q}{n[P] \to_{\mathcal{R}} n[Q]}$$

(Par)
$$\frac{P \to_{\mathcal{R}} Q}{P \mid R \to_{\mathcal{R}} Q \mid R}$$

(Congr)
$$\frac{P \equiv P' \wedge P' \to_{\mathcal{R}} Q' \wedge Q' \equiv Q}{P \to_{\mathcal{R}} Q}$$

(In)
$$m[\mathrm{in}\ n.P \mid Q] \mid n[R] \to_{\mathcal{R}} n[m[P \mid Q] \mid R]$$
where $m \in \mathcal{C}_P$

(out)
$$n[m[\mathrm{out}\ n.P \mid Q] \mid R] \to_{\mathcal{R}} m[P \mid Q] \mid n[R]$$
where $m \in \mathcal{C}_P$

Table 2-5: **Transition relation (1);** $P \to_{\mathcal{R}} P'$**; parameterized on** $\mathcal{R}$**.**

As the tables present, there are two variants of reduction relation $\to_{\mathcal{R}}$ where $\mathcal{R}$ identifies two instantiations of the relation: One variant ($\to_{RM}$) takes care of annotations, the other ($\to$) omits them. In both instantiations the reduction relation is the least relation on closed processes i.e. processes without free variables. The differences between the two instances are described below:

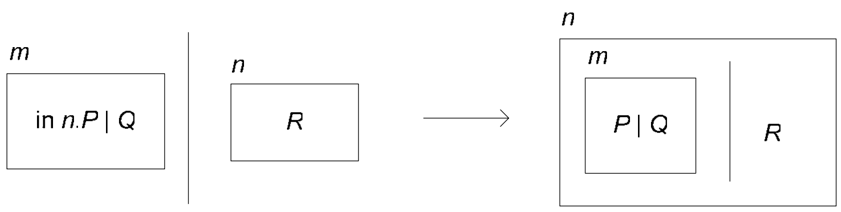
- The standard semantics written $P \to Q$ will treat $\mathcal{R}$ to be universally true.

- The *reference monitor semantics* written $P \rightarrow_{RM} Q$ takes RM(ℓ, $\mathcal{L}'$, ℓ', $\mathcal{L}$) = ($\ell \in \mathcal{L}' \wedge \ell' \in \mathcal{L}$). By this way we require that decryptions may only happen at the crypto-points designated when the corresponding encryptions were made and vice-versa. Otherwise the execution is aborted.

The rules in Table 2-5 specify all reduction rules which are not communication-related. The rule (Resn) expresses that the restriction construct of private names is transparent with respect to reduction.

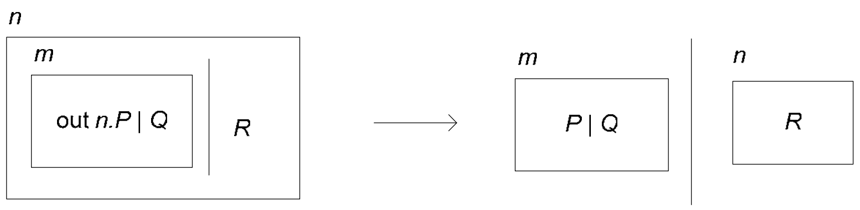
The reduction rule (Amb) specifies capabilities nested inside ambients may be actively running if they are not prefixed with other capabilities. The inference rule (Par) reduces on the left branch; reduction on the right part can be obtained by commutativity given by structure congruence. The rule (Congr) allows us using equivalence during reduction.

The in-capability navigates the enclosing ambient to move into some sibling ambient running in parallel. This can be depicted as:



If successful, the reduction reorganizes a sibling *n* of the ambient *m* to be a child ambient of *n*. After the execution, the process "in *n*.*P*" continues with *P* and both the two parallel processes *P* and *Q* are at a lower level in the tree of ambients.

The out-capability instructs the ambients surrounding the process "out *m*.*P*" to move out of its parent ambient. That is illustrated as:



We shall write $P\{x \leftarrow M\}$ for the substitution of the capability or name for each free occurrence of the free variable *x* in the process *P*. Usually we omit trailing inactive process by writing *M* for *M*.0.

Example 2.2 Following Example 2.1, the standard semantics of the process is illustrated as:

$$\begin{array}{ll} A[p[\text{out } A.\text{in } B]] \mid B[\ ] & \\ \rightarrow A[] \mid p[\text{in } B] \mid B[\ ] & \text{(Out)} \\ \rightarrow A[] \mid \ \mid B[p[]] & \text{(In)} \end{array}$$

by the rules (Out) and (In). □

(LocCom)

$$\frac{\wedge_{i=1}^{j} M_i = M_i' \quad \wedge \quad \mathcal{R}(\ell, \mathcal{D}, \varepsilon, \mathcal{L})}{<M_1,\cdots,M_k>^{\circ}_{\ell}[\text{dest } \mathcal{L}] \mid (M_1',\cdots,M_j';x_{j+1},\cdots,x_k)^{\circ}.P \rightarrow_{\mathcal{R}} P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\}}$$

(Output-Chd1)

$$\frac{\wedge_{i=1}^{j} M_i = M_i'}{<M_1,\cdots,M_k>^{n} \mid n[(M_1',\cdots,M_j';x_{j+1},\cdots,x_k)^{\circ}.P \mid Q] \rightarrow_{\mathcal{R}} n[P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid Q]}$$

(Output-Chd2)

$$\frac{\wedge_{i=1}^{j} M_i = M_i' \quad \wedge \quad \mathcal{R}(\ell, \mathcal{D}, \varepsilon, \mathcal{L})}{<M_1,\cdots,M_k>^{\circ}_{\ell}[\text{dest } \mathcal{L}] \mid n[(M_1',\cdots,M_j';x_{j+1},\cdots,x_k)^{\uparrow}.P \mid Q] \rightarrow_{\mathcal{R}} n[P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid Q]}$$

(Output-Par1)

$$\frac{\wedge_{i=1}^{j} M_i = M_i'}{(M_1',\cdots,M_j';x_{j+1},\cdots,x_k)^{\circ}.P \mid n[<M_1,\cdots,M_k>^{\uparrow} \mid R] \rightarrow_{\mathcal{R}} P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid n[R]}$$

(Output-Par2)

$$\frac{\wedge_{i=1}^{j} M_i = M_i' \quad \wedge \quad \mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})}{(M_1',\cdots,M_j';x_{j+1},\cdots,x_k)^{n}_{\ell'}[\text{orig } \mathcal{L}'].P \mid n[<M_1,\cdots,M_k>^{\circ}_{\ell}[\text{dest } \mathcal{L}] \mid R] \rightarrow_{\mathcal{R}} P\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid n[R]}$$

Table 2-6: **Transition relation (2); $P \rightarrow_{\mathcal{R}} P'$; parameterized on $\mathcal{R}$.**

Five rules for communication primitives are specified in Table 2-6. We first explain their standard semantics in which all annotations can be removed and $\mathcal{R}$ is always true. To simplify our presentation we illustrate them in their monadic version: Only one message is communicated at a time and thus no matching is needed. One can easily extend them to the polyadic ones. Additionally as we shall only apply the transition relation to closed processes, all message of output $<M_1,\cdots,M_k>^{\eta}$ is variable free, saying that $\text{fv}(<M_1,\cdots,M_k>^{\eta}) = \varnothing$.

The first rule is for local communication that happens between any two sibling ambients, i.e.

$$<M>^{\circ} \;\Big|\; (;x)^{\circ}.P \longrightarrow P\{x \leftarrow M\}$$

and binds the variable x with the message M in the receiving process as no matching is needed in the input. There are two rules for output to child ambients: One is the enclosing ambient explicitly uses the child's mailbox to exchange message as below figure shows.

$$<M>^{n} \;\Big|\; \overset{n}{\boxed{(;x)^{\circ}.P \mid Q}} \longrightarrow \overset{n}{\boxed{P\{x \leftarrow M\} \mid Q}}$$

Another way is to make use of the enclosing ambients mailbox to exchange message

$$<M>^{\circ} \;\Big|\; \overset{n}{\boxed{(;x)^{\uparrow}.P \mid Q}} \longrightarrow \overset{n}{\boxed{P\{x \leftarrow M\} \mid Q}}$$

The last two rules is for output to a parent

$$(;x)^{\circ}.P \;\Big|\; \overset{n}{\boxed{<M>^{\uparrow} \mid R}} \longrightarrow P\{x \leftarrow M\} \;\Big|\; \overset{n}{\boxed{R}}$$

and

$$(;x)^{n}.P \;\Big|\; \overset{n}{\boxed{<M>^{\circ} \mid R}} \longrightarrow P\{x \leftarrow M\} \;\Big|\; \overset{n}{\boxed{R}}$$

Particularly, according to the transition relation in Table 2-6, output for a grandchild is not allowed, such as:

$$< M >^{c} \mid c[g[(;x)^{\uparrow}.P] \mid Q] \not\rightarrow \cdots$$

But one can take advantage of the child to forward a message to the grandchild as below

$$< M >^{c} \mid c[(;x)^{\circ}.<;x>^{\circ} \mid g[(;x)^{\uparrow}.P] \mid Q]$$

Now we take matching into account. All rules in Table 2-6 expresses that an output $< M_1,\cdots,M_j,M_{j+1},\cdots,M_k >$ is matched by an input $(M_1',\cdots,M_j';x_{j+1},\cdots,x_k)$ if the first j elements are pairwise the same. Otherwise no variable binding occurs.

Last we consider the reference monitor semantics in which the annotated labels are concerned with; it means we need consider the condition RM(ℓ, $\mathcal{L}'$, ℓ', $\mathcal{L}$) for the rules (LocCom), (Output-Chd2) and (Output-Par2).

Besides matching, the rules (LocCom) and (Output-Chd2) specify that RM(ℓ, $\mathcal{D}$, ε, $\mathcal{L}$) is true if the destination set $\mathcal{L}$ includes the trivial crypto-point ε. Here the check of the pair $(\ell, \mathcal{D})$ is trivially true.

For the rule (Output-Par2) it expresses that the message contained by the child ambient and encrypted with the name *n* is decrypted by the enclosing ambient. When successful, each M_i is bound to the corresponding x_i. Furthermore the reference monitor ensures that the crypto-point of the encrypted message is acceptable at the decryption, i.e. $\ell \in \mathcal{L}'$, and the crypto-point of the decryption is acceptable for the encryption, i.e. $\wedge\ \ell' \in \mathcal{L}$. In the standard semantics the condition $\mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})$ is universally true and thus can be ignored.

The reason that we check the condition RM(ℓ, $\mathcal{L}'$, ℓ', $\mathcal{L}$) in the reference monitor semantics in the rules (LocCom) and (Output-Chd2) is because there is not an explicit primitive for encryptions or decryptions in ambient and the local output primitive is used for the communication of the same as well as adjacent layer(s) (i.e. parent or child ambient(s)). With the help of ε, we can take advantage of labels to distinguish an encrypted message from a plain one. For example, a message is plain if the local output is in the form of $< \mathrm{M}_1, \cdots, \mathrm{M}_k >^{\circ}_{\varepsilon} [\mathrm{dest}\ \mathcal{D}]$. For any encrypted message, the crypto-point must not be ε. Similarly if an input from a child is supposed to model a decryption, we use labels other than ε. Otherwise the crypto-point of the input is ε and the message received is supposed to be plain.

Example 2.3 Based on the process presented in Example 2.1, we let the packet contain two messages, a public message m_1 and a secret message m_2. This could be programmed as

$$A[p[\mathrm{out}\ A.\mathrm{in}\ B.(< m_1 >^{\uparrow} | < m_2 >^{\circ}_{\mathrm{A}} [\mathrm{dest\ B}])]]\ |$$
$$B[(; x_1)^{\circ}.P \,|\, (; x_2)^{p}_{\mathrm{B}}[\mathrm{orig\ A}].P']$$

Using child-to-parent output, we mean that m_1 is public-accessed by any enclosing ambient. But to get m_2 the enclosing ambient must know the name *p* at first. In that sense we say m_2 is secret. Also according to the syntax, we add annotations for the local output and parent-child input. The standard semantics of the process is then presented as below.

$$\begin{aligned}
& A[p[\mathrm{out}\ A.\mathrm{in}\ B.(< m_1 >^{\uparrow} | < m_2 >^{\circ}_{\mathrm{A}} [\mathrm{dest\ B}])]]\ |\ B[(; x_1)^{\circ}.P \,|\, (; x_2)^{p}_{\mathrm{B}}[\mathrm{orig\ A}].P'] \\
\to\ & A[]\ |\ p[\mathrm{in}\ B.(< m_1 >^{\uparrow} | < m_2 >^{\circ}_{\mathrm{A}} [\mathrm{dest\ B}])]\ |\ B[(; x_1)^{\circ}.P \,|\, (; x_2)^{p}_{\mathrm{B}}[\mathrm{orig\ A}].P'] \\
\to\ & A[]\ |\ B[p[< m_1 >^{\uparrow} | < m_2 >^{\circ}_{\mathrm{A}} [\mathrm{dest\ B}]] \,|\, (; x_1)^{\circ}.P \,|\, (; x_2)^{p}_{\mathrm{B}}[\mathrm{orig\ A}].P'] \\
\to\ & A[]\ |\ B[p[< m_2 >^{\circ}_{\mathrm{A}} [\mathrm{dest\ B}]] \,|\, P\{x_1 \leftarrow m_1\} \,|\, (; x_2)^{p}_{\mathrm{B}}[\mathrm{orig\ A}].P'] \\
\to\ & A[]\ |\ B[p[] \,|\, P\{x_1 \leftarrow m_1\} \,|\, P'\{x_2 \leftarrow m_2\}]
\end{aligned}$$

□

2.4 Assertions for the Origin and Destination

In the syntax of our calculus, we only annotate local output and input (from child ambients). Such annotations should be attached to the program point where encryptions and decryptions occur. In Lysa calculus, there are explicit cryptographic primitives for these two operations. For Boxed Ambients, however, they are implicit. Thus we investigate if there are other possibilities for expressing encryptions and decryptions in Boxed Ambients. First we investigate how encryptions and decryptions can be expressed in Boxed Ambients. As far as we can see, there are two ways to express encryptions and decryptions:

(1) ***Encoding a message in an anonymous packet.*** Typically a packet consists of two kinds of information: (1) public names which is just plain text and (2) secret names which are encrypted by some key *K*. The first way to formalize this kind of packets in Boxed Ambients is to use an anonymous packet as:

$$p[\text{out } A.\text{in } B.(<public>^{\uparrow} \mid K[\text{out } p.<secret>^{\circ}])]$$

Once the capabilities "out *A*" and "in *B*" have been executed the enclosing ambient could have access to the public parts of the package. Here "anonymous" means that the enclosing ambient do not have to know the name of the packet when it reads the contents of the packet. This could be illustrated by following process evolvement.

$$\begin{aligned} & p[(< public >^{\uparrow} \mid K[out\ p.< \sec ret >^{\circ}])] \mid (x)^{\circ}.x \\ \rightarrow\ & p[K[out\ p.< \sec ret >^{\circ}]] \mid public \\ \rightarrow\ & p[] \mid K[< \sec ret >^{\circ}] \mid public \end{aligned}$$

Please note that in order to decrypt the secret parts of the message the enclosing ambient needs knowledge of the key *K*. That can be illustrated as:

$$K[< \sec ret >^{\circ}] \mid (x)^{K}.x \rightarrow K[] \mid \sec ret$$

(2) ***Reusing the cryptographic key K as packet name*** A more concise way to encode a packet is to reuse the cryptographic key *K* and take advantage of the primitive of output communication with enclosing ambients as:

$$K[\text{out } A.\text{in } B.(< public >^{\uparrow} \mid < \sec ret >^{\circ})]$$

After the capabilities out A and in B have been executed, the public part of *K* is still accessible to the enclosing ambient without the key *K*, e.g.

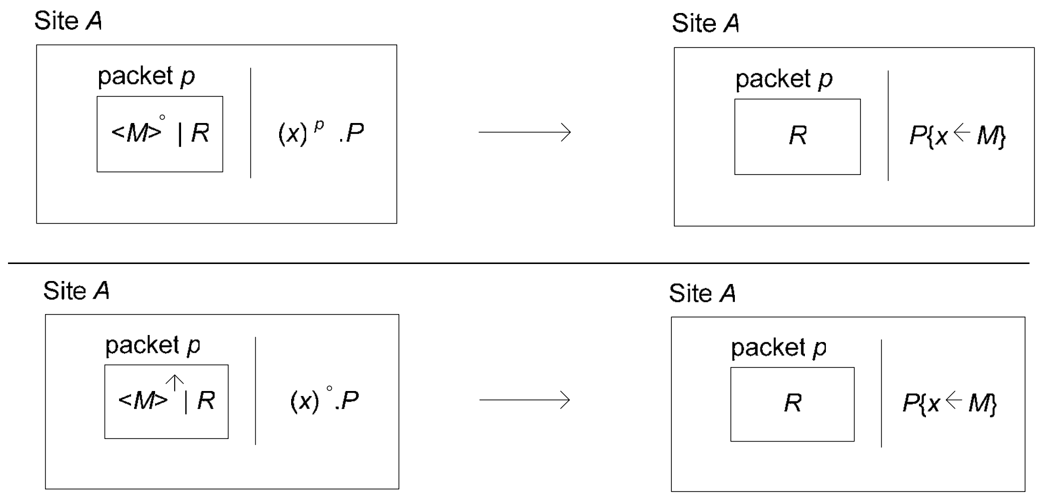
$$K[< public >^{\uparrow} \mid < \sec ret >^{\circ}] \mid (x)^{\circ}.x \rightarrow K[< \sec ret >^{\circ}] \mid public$$

However to get hold of the secret parts of the packet the enclosing ambient still needs to know the key *K* before it can access to the secret names, e.g.

$$K[< \sec ret >^{\circ}] \mid (x)^{K}.x \rightarrow K[] \mid \sec ret$$

Summarizing the above two ways of encoding a message, we find that the encryptions are modeled by nesting local output communication primitive into a packet ambient. Decryptions are expressed with child-parent input between the ambients of packets and sites. But before we reach final conclusion that where the annotation should be added, let us check whether or not other communication primitives could be used to express encryptions and decryptions.

Suppose the ambients of sites never turn up inside the ambients of packets, we search for which primitive(s) could be used for modeling encryptions and decryptions. As we have not open-capability, child-to-parent communication is the only way a site could use to get message from a packet. Among the six communication primitives, this could be implemented by two pairs of communication primitives depicted as the below figure shows:



In the upper figure the enclosing ambient must know the name *p* to retrieve the mailbox of the packet *p*. In the case shown at the lower part of the figure, however, the enclosing ambient can always get the message sent by the packet *p* and does not have to know the name *p*.

Since we have two pairs of primitives to express child-to-parent communication, could both of them be proper to present encryptions and decryptions? As we have presented at the beginning of this section, the pair of local output, such as $< M >^{\circ}$, and input from child, such as $(x)^{p}$, can express encryptions and decryptions, we here focus on analyzing the output primitive $< M >^{\uparrow}$ and input primitive $(x)^{\circ}$. By the nature of $< M >^{\uparrow}$, we can not control which site shall receive the message so that the message M is always public to any site enclosing the packet. Therefore we can only adopt the output $< M >^{\circ}$ to transfer secret names in a packet. Correspondingly the input message $(x)^{\circ}$ can not be used to read the message from its enclosed packets. Thus we reach our conclusion: Only the pair local output and child-parent input can be used to model encryption and decryption in Boxed Ambients. And we are sure that only these two communication primitives need annotating with labels.

Until now we only consider annotating the communication primitives. But are there other places for annotating the orig and dest? For example, could we annotate an ambient? The answer to the question lies in that to which part of a packet authentication intention should be checked, i.e. to the whole packet or to the encrypted part. Our answer is the later one. To be illustrative we consider following process annotated on ambients:

$$K_{AS}[\text{out } A.\ \text{in } S.\ (< A >^{\uparrow} | < B,\ K_{AB} >^{\circ})]^{A}[\text{dest } S]$$

Assume the packet is cached by an attacker *M* and then the destination *S* will be different from the decryption point. However we could not specify that any authentication intention is violated before the attacker could decrypt the secret parts of the message. Thus annotating two communication primitives will give us better approximation in analysis.

If one prefers to annotate ambient then a possible choice is the original **Mobile Ambients** [6,5]. With Mobile Ambients we could take advantage of open-capability to model decryption, and then we do not rely on the communication primitives between adjacent layers of ambients. The encryption is to enclose secret names into an ambient named with the secret key *K*

$$K[< \sec ret >]$$

The message encoded in anonymous packets is

$$p[\text{out } A.\text{in } B.(< public >|\ K[< \sec ret >])]$$

After the packet moves into the ambient *B*, the enclosing ambient gets public and secret names by opening the ambients *p* and *K* sequentially and then communicating with their local box. The evolvement of the process is illustrated below.

$$\text{p[<public>| K[<secret>]] | open p.}(x_p)\text{.open K.}(x_K)\text{.P} \rightarrow$$
$$\text{<public>| K[<secret>] | }(x_p)\text{.open K.}(x_K)\text{.P} \rightarrow$$
$$\text{K[<secret>] | open K.}(x_K)\text{.P}\{x_p \leftarrow \text{public}\} \rightarrow$$
$$\text{<secret> | }(x_K)\text{.P}\{x_p \leftarrow \text{public}\} \rightarrow$$
$$\text{P}\{x_p \leftarrow \text{public},\ x_K \leftarrow \text{secret}\}$$

Since opening ambients are used to model encryption and decryption we could annotate ambients to check authentication intentions. As the [9, 8] states that open-capability is somewhat coarse in allowing all contents of an enclosed ambient to be resolved. We here would like to base our calculus on Boxed Ambients and will not discuss the effects of introducing "open" capability further in this thesis.

Summarizing the above discussion, we are sure that we can annotate local output and child-parent input only in our syntax as:

$$<M_1,\ ...,\ M_k>^{\circ}_{\ell}\ [\text{dest } \mathcal{L}]$$
$$(M'_1,\ ...,\ M'_j;\ x_{j+1},\ ...,\ x_k)^{n}_{\ell}[\text{orig } \mathcal{L}]$$

2.5 Modeling of Hierarchical Networks

To model a hierarchical network, basically our calculus should be able to represent the concepts of networks including the public-accessed network and local networks, and participants of a protocol. In this subsection we address the formalism of these concepts. They are important in both directing the formalization of a protocol and defining the attacker.

To program a hierarchical network in ABoxed Ambients calculus, the program of interest is the ambient of the form $n_\star[P_\star]$ where $n_\star \notin \mathrm{fn}(P_\star)$, and where $P_\star$ satisfies the conjunction of the following conditions

- $P_\star$ is closed; formally $\mathrm{fv}(P_\star) = \emptyset$.
- $P_\star$ is well-formed with respect to the well-formedness $\Gamma \vdash wf_s(P_\star)$.

Here the type environment is given by

$$\Gamma : \mathcal{N} \to \mathcal{P}(\mathcal{C} \cup M)$$

which maps a constant to itself or maps a variable to its possible values. The well-formedness predicate $\Gamma \vdash wf_s(P_\star)$ (see Table 2-7) serves five purposes: (1) Declaring that site ambients can not move. (2) Imposing the conditions that the name of a site and a packet ambient have the type of $\mathcal{S}$ and $\mathcal{P}$ respectively. (3) Enforcing attacker processes act just like sites. (4) Allowing site ambients to contain both site ambients and packet ambients which is then defined by the well-formedness $\Gamma \vdash wf_p(P_\star)$. (5) Disallowing a site read its enclosing ambient's mailbox by child-parent input $(x)^\uparrow$.

$\Gamma \vdash wf_S(0)$	$\dfrac{\Gamma \vdash wf_S(P)}{\Gamma \vdash wf_S((\nu\ C)P)}$	$\dfrac{\Gamma \vdash wf_S(P)}{\Gamma \vdash wf_S((\nu k\ n)P)}$	$\dfrac{\Gamma \vdash wf_S(P)}{\Gamma \vdash wf_S(!P)}$
$\dfrac{\Gamma \vdash wf_S(P_1)\ \ \Gamma \vdash wf_S(P_2)}{\Gamma \vdash wf_S(P_1 \mid P_2)}$	$\dfrac{\Gamma \vdash N : \mathcal{S}\ \ \Gamma \vdash wf_S(P)}{\Gamma \vdash wf_S(N[P])}$ if $\Gamma(N) \subseteq \mathcal{C}_s$	$\dfrac{\Gamma \vdash N : \mathcal{P}\ \ \Gamma \vdash wf_P(P)}{\Gamma \vdash wf_S(N[P])}$ if $\Gamma(N) \subseteq \mathcal{C}_p$	$\Gamma \vdash wf_S(<M_1, \cdots, M_k>^\eta)$
$\dfrac{\Gamma \vdash wf_S(P)}{\Gamma \vdash wf_S((M'_1, \cdots, M'_j; x_{j+1}, \cdots, x_k)^\eta.P)}$ where $\eta \neq \uparrow$	$\Gamma \vdash wf_S(\bullet)$		

Table 2-7: Well-formedness of site processes with respect to Γ : $\Gamma \vdash wf_s(P_\star)$

The predicate $\Gamma \vdash wf_p(P_\star)$ defined on the next page serves four purposes: (1) Stating that packet ambients is movable. That is the major difference between site ambients and packet ambients. (2) Imposing that the name of a packet ambient has type $\mathcal{P}$. (3) Enforcing the condition that packet ambients can only contain packet ambients and (4) that packet ambients do not have input primitives.

$$\begin{array}{cccc}
\Gamma \vdash wf_p(0) & \dfrac{\Gamma \vdash wf_p(P_1) \;\; \Gamma \vdash wf_p(P_1)}{\Gamma \vdash wf_p(P_1 \mid P_2)} & \dfrac{\Gamma \vdash wf_p(P)}{\Gamma \vdash wf_p(!P)} & \dfrac{\Gamma \vdash wf_p(P)}{\Gamma \vdash wf_p(M.P)} \\[2ex]
 & \dfrac{\Gamma \vdash wf_p(P)}{\Gamma \vdash wf_p((\nu k\ n)P)} & \dfrac{\Gamma \vdash wf_p(P)}{\Gamma \vdash wf_p((\nu\ N)P)} & \begin{array}{c}\dfrac{\Gamma \vdash N : \mathcal{P} \;\; \Gamma \vdash wf_p(P)}{\Gamma \vdash wf_p(N[P])} \\ \text{if}\, \Gamma(N) \subseteq \mathcal{C}_p\end{array} \\[2ex]
\Gamma \vdash wf_p(<M_1, \cdots, M_k>^{\eta}) & \dfrac{\Gamma \vdash wf_p(P)}{\Gamma \vdash wf_p((M'_1, \cdots, M'_j; x_{j+1}, \cdots, x_k)^{\eta}.P)} & &
\end{array}$$

Table 2-8: Well-formedness of packet processes with respect to Γ : $\Gamma \vdash wf_p(P_\star)$

Under most cases we focus our discussion on the process $P_\star$ while the $n_\star$ becomes implicitly assumed.

2.6 The ABoxed Ambients Specification

Cryptographic protocols usually involve several roles, such as server *S*, participants *A* (initiator) and *B* (responder). It is quite normal for a server to look up a key in its key table for the corresponding agents. With ambients we can model the key storage and retrieving in a quite natural way. This idea is first introduced in [3] and illustrated as below.

We can have a process of the form

$$\text{KeyTable} \;=\; \textit{datafile}[!<n_1, K_1>^{\circ}_{\varepsilon} [\text{dest}\, \mathcal{D}]] \mid \cdots \mid !<n_m, K_m>^{\circ}_{\varepsilon} [\text{dest}\, \mathcal{D}]]$$

where we model a data file on the server with an ambient named *datafile*. The name of agents and its private key K_i stored in the file are modeled by local output. The replication (!) is necessary to present that the key table is persistent and can be queried any number of times. The trivial crypto-point ε and the destination $\mathcal{D}$ imply that the file are not supposed to be encrypted. Correspondingly looking up key table is in the form of

$$\text{Keytable} \mid (n_i; y_k)^{\text{datafile}}_{\varepsilon} [\text{dest}\, \mathcal{D}]. \cdots .y_k \cdots$$

The name of the agent n_i is first checked in input. If success, the private key of the agent n_i will be bound with the variable y_k and used in the following process.

We sometimes need to test if two numbers or names are equal before proceeding. In Mobile Ambients this can be coded by creating an ambient, *n*[ ] and then let an "open" capability, i.e. open *m*, to block its following process *P* as

n[ ] | open *m.P*

In Boxed Ambients there is not "open" capability and then communication is used: an ambient, $n[<>^{\circ}]$ performs a local nullary output and make use of an input, $()^m$, to block its following processes

$$n[<>^{\circ}] \mid ()^{m}.\, P$$

However the guard condition can be coded in a quite natural and concise way in ABoxed Ambients. We simply take advantage of the matching of input and output as:

$$<n>^{\circ} \mid (m;)^{\circ}.\, P$$

By this way we do not need to introduce a new ambient.

Wide Mouthed Frog We consider the following version of the Wide Mouthed Frog protocol [31] where two agents A and B share master keys K_{AS} and K_{BS} respectively with a trusted server S. The protocol aims at first exchanging a secret session key K_{AB} for the use between A and B, and then communicating secret message M with the session key. The protocol narration is

1. $A \to S:\ A, K_{AS}[B, K_{AB}]$
2. $S \to B:\ K_{BS}[A, K_{AB}]$
3. $A \to B:\ K_{AB}[M]$

where we write $K[M]$ for the message M encrypted under the key K.
This kind of protocol narration is informal and some important information is unclear or missing when we need program a protocol including that (1) the outputs and inputs between participants are not distinguished, (2) the encryption and decryption are unclear, (3) fresh keys and key-checking in a protocol are implicit and specifically (4) the authentication intentions and origins are missing while they are needed to validate the authenticity of a protocol. To make them clear, we borrow the idea of [2] in declaring an extended narration for the corresponding protocol. The narration of WMF is then expanded as below.

$$\begin{array}{lrcl}
1. & A \to & : & A, K_{AS}[B, K_{AB}][\text{dest S}] \quad \{\text{assuming } K_{AB} \text{ is a new key}\} \\
 & \to S & : & x_A, x \qquad\qquad \{\text{check } x_A = A\} \\
 & S & : & \text{decrypt } x \text{ as } K_{AS}[x_B, x_{K_{AB}}][\text{orig A}] \quad \{\text{check } x_B = B\} \\
2. & S \to & : & x_B[x_A, x_{K_{AB}}][\text{dest B}] \\
 & \to B & : & y \\
 & B & : & \text{decrypt } y \text{ as } K_{BS}[y_A, y_{K_{AB}}][\text{orig S}] \quad \{\text{check } y_A = A\} \\
3. & A \to & : & K_{AB}[M][\text{dest B}] \\
 & \to B & : & z \\
 & B & : & \text{decrypt } z \text{ as } y_{K_{AB}}[z_M][\text{orig A}]
\end{array}$$

For now we assume all participants show up at the same network and later on we shall consider more complex cases. As presented before, the KeyTable to be used by S is in the form of

$$\text{KeyTable} = \mathit{datafile}[!<A, K_{AS}>^{\circ}_{\varepsilon}[\text{dest } \mathcal{D}] \| !<B, K_{BS}>^{\circ}_{\varepsilon}[\text{dest } \mathcal{D}]]$$

The protocol can be formalized in ABoxed Ambients as:

$$
\begin{array}{l}
(\nu\ K_{AS})(\nu\ K_{BS}) \\
(A[(\nu\ K_{AB})\ K_{AS}[\text{out } A.\text{ in } S.\ (<A>^{\uparrow} \mid < B, K_{AB} >^{\circ}_{\mathrm{A}_1}\ [\text{dest } \mathrm{S}_1])] \mid \\
\qquad\qquad (\nu\ M)\ K_{AB}[\text{out } A.\text{ in } B.\ <M>^{\circ}_{\mathrm{A}_2}[\text{dest } \mathrm{B}_2]] \\
\mid \\
S[\text{KeyTable} \mid \\
\quad (A;)^{\circ}.(A; y_{K_{AS}})^{\text{datafile}}_{\varepsilon}[\text{dest } \mathcal{D}].(B;\ \mathrm{y}_{K_{AB}})^{\mathrm{y}_{K_{AS}}}_{\mathrm{S}_1}[\text{orig } \mathrm{A}_1]. \\
\qquad\qquad (B;\mathrm{y}_{K_{BS}})^{\text{datafile}}_{\varepsilon}[\text{dest } \mathcal{D}].\mathrm{y}_{K_{BS}}[\text{out } S.\text{ in } B.< A,\ \mathrm{y}_{K_{AB}} >^{\circ}_{\mathrm{S}_2}[\text{dest } \mathrm{B}_1]] \\
\mid \\
B[(A;\ \mathrm{z}_{K_{AB}})^{K_{BS}}_{\mathrm{B}_1}[\text{orig } \mathrm{S}_2].(;z)^{\mathrm{z}_{K_{AB}}}_{\mathrm{B}_2}[\text{orig } \mathrm{A}_2]])
\end{array}
$$

Table 2-9: WMF in ABoxed Ambients

We explain it step by step:

1. A generates a new session key K_{AB} by restriction construct

   $(\nu\ K_{AB})$

   and then sends the message to the server S. As the new session key K_{AB} is encrypted by A's master key K_{AS}, we annotate the local output as

   $K_{AS}[\text{out } A.\text{ in } S.\ (<A>^{\uparrow} \mid < B, K_{\mathrm{AB}} >^{\circ}_{\mathrm{A}_1}\ [\text{dest } \mathrm{S}_1])]$

   Once the capabilities out A.in S have been performed, the agent name A can be received by any enclosing ambient. But B's name and the session key $(B,\ K_{AS})$ can only be read by the enclosing ambients knowing the master key K_{AS}.

2. First, the server receives A's name and query her master key from KeyTable. Then she decrypt the encrypted part ($\mathrm{y}_B = B$ and $\mathrm{y}_{K_{AB}} = K_{AB}$) of the packets where the reference monitor will check the authentication intentions with the fixed labels:

   $\text{KeyTable} \mid (A;)^{\circ}.(A; y_{K_{AS}})^{\text{datafile}}_{\varepsilon}[\text{dest } \mathcal{D}].(B;\ \mathrm{y}_{K_{AB}})^{\mathrm{y}_{K_{AS}}}_{\mathrm{S}_1}[\text{orig } \mathrm{A}_1]$

   Second the server queries B's master key ($\mathrm{y}_{\mathrm{K}_{BS}} = K_{BS}$) for KeyTable, then encrypts the name A and the session key ($\mathrm{y}_{K_{AB}} = K_{AB}$) with B's master key and send it to B ($\mathrm{y}_B = A$):

   $(B;\mathrm{y}_{K_{BS}})^{\text{datafile}}_{\varepsilon}[\text{dest } \mathcal{D}].\mathrm{y}_{K_{BS}}[\text{out } S.\text{ in } \mathrm{y}_B.< \mathrm{y}_A,\ \mathrm{y}_{K_{AB}} >^{\circ}_{\mathrm{S}_2}[\text{dest } \mathrm{B}_1]]$

   Again we add annotations at local output as the encryption occurs there.

   The agent B receiving message from the server S decrypts the message with his master key K_{BS}:

$(A;\, z_{K_{AB}})_{B_1}^{K_{BS}}[\text{orig } S_2]$

3. Last the agent *A* sends the message *M* encrypted with the new session key K_{AB} to *B*

   $(\nu\, M)\, K_{AB}[\text{out } A.\ \text{in } B.\ \langle M \rangle_{A_2}^{\circ}[\text{dest } B_2]]$

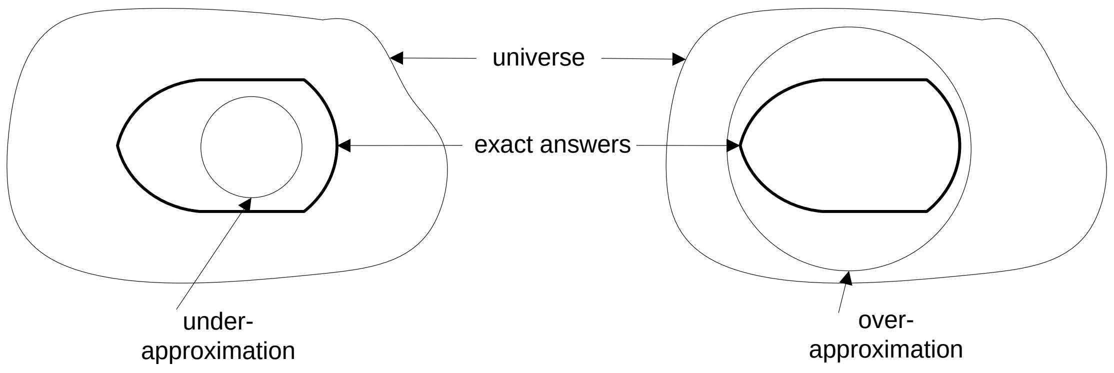
   B receives and decrypts it with the session key ($z_{K_{AB}} = K_{AB}$) and the reference monitor will verify the authentication intention for the decrypted message:

   $(;x)_{B_2}^{z_{K_{AB}}}[\text{orig } A_2]$

   This completes the programming of the protocol.

3 Control flow analysis

The analysis approach used in this thesis originates from a technique called *static analysis*. It is a long-history-studied field which was developed to optimize compilers at the beginning. The technique is called static because it calculates some aspect of the property of a program statically at compile-time without actually running the program. To keep it computable static analysis focuses on calculating approximation to certain program behavior rather than exact answers. Basically we need guarantee our approximation to be safe with respect to a formal semantics. There are two classes of approximation compared with exact answers: over-approximation and under-approximation. An over-approximation covers all exact answers while an under-approximation is strictly covered by exact answers. We present the idea graphically in below diagram.



It is important to be clear which kind of approximation are needed when a specific question is asked. Usually a question asking ‘what must happen’ requires under-approximation and a question asking ‘what may happen’ implies over-approximation. We could then divide the corresponding static analysis into *may analysis* and *must analysis*. Our analysis thus belongs to the group of *may analysis* (over-approximation) because we take care of all decryptions that may happen in a cryptographic protocol.

We could also classify analyses according to whether they calculate data flow or control flow. However when process calculi are considered, it becomes not easy to distinguish between data and program control structures due to their succinct and expressive nature. Like the analysis for LySa-calculus [2], mobile ambients [3], etc. we consider our analysis a kind of *control flow analysis*.

There are several classic approaches to *static analysis*, such as Data Flow Analysis [15], Constraint Based Analysis [16], Abstract Interpretation [17], and Type and Effect Systems [18], etc. A thorough and good introduction to them could be found at [1].

Flow Logic is a relatively new technique in static analysis field compared with above analyses. But still its development is mainly based on three existing technologies, Data Flow Analysis, Constraint Based Analysis and Abstract Interpretation. The specification of an analysis in Flow Logic declares a set of constraints that analysis estimation must satisfy in order to be acceptable to a program. Thus a flow logic specification focuses on not how the analysis is computed but what the analysis does. By this way specification and implementation give rise to be independent and an analysis designer can then concentrate on specifying the analysis and not worry about its design and implementation at the same time. That is why we choose Flow Logic to specify our control flow analysis.

The presentation of this chapter will be in three parts: First we describe some general concepts of Flow Logic that will be used in the control flow analysis of ABoxed Ambients. Then the specification of the analysis is given and explained. Last a short summary reviews the topics discussed in this chapter.

3.1 Flow Logic in ABoxed Ambients

Flow logic, which is based on logical systems, is a form of formalism for static analysis. It was first developed in [19] for a pure functional language, $\lambda -$calculus. After that this technique has been applied on many calculi such as Ambient Calculus [10], spi-calculus [37], and so on. A good tutorial on the approach could be found at [14].

A flow logic specification consists of three parts:

(1) The universe of discourse for the analysis estimates; usually the universe is given by complete lattices;

(2) The format of the judgements;

(3) The defining clauses.

This high abstraction clearly separates analysis specification from its implementation and thus facilitates analysis design. In Flow Logic we normally define a predicate in the format of

$$\mathcal{A} \models P$$

which holds when $\mathcal{A}$ is an acceptable approximation of the interested property of the process P. Then the analysis is defined by a well-defined predicate which, in this thesis, will be the form

$$\mathcal{A} \models P \;\; \textbf{iff} \;\; \mathcal{A} \models P' \text{ and logic formulas specifying constraints } \mathcal{A} \text{ should follow}$$

where P' is always the subprocesses of P. Therefore our flow logic specification is inductively defined. In Flow Logic this kind of specifications are called *compositional* since the analysis of a process is only relevant to the analysis of its part.

Apparently there may be many acceptable approximations $\mathcal{A}$ for a process P. Fortunately flow logic specifications enjoy the Moore family property that guarantees the existence of a best or most precise analysis result.

3.1.1 Semantic Correctness of the Ananlysis

In Chapter 2 we gave a reduction semantics for ABoxed Ambients and therefore the behaviors of an ambient process can be described by a series of reduction steps that the process can make. The semantic correctness of the analysis should then be established in terms of whether the analysis follows the semantics in a reasonable way.

As we mentioned at the beginning of this chapter our analysis will make an over-approximation that means the analysis must strictly over approximate the behavior of the given semantics. This is normally easy to check or prove by reviewing the specification (that is also a benefit given by Flow Logic).

On the other hand the static nature of control flow analysis requires that the analysis domain has enough information about the whole execution of a process. For the induction semantics this can be formally proved by stating that

$$\text{If } \mathcal{A} \models P \text{ and } P \to Q \text{ then } \mathcal{A} \models Q$$

It states that if the analysis $\mathcal{A}$ is an acceptable analysis result for P and P may evolve to Q then $\mathcal{A}$ is also an acceptable analysis result for Q. Next we prove that

$$\text{If } \mathcal{A} \models P \text{ and } P \to^{*} Q \text{ then } \mathcal{A} \models Q$$

Here * stands for the transitive and reflexive closure of the relation. Since Q is any process P can evolve to, we are sure that $\mathcal{A}$ indeed has all necessary information to be an acceptable analysis result for the entire execution of P.

3.1.2 Succinct versus Verbose Flow Logics

The form of flow logic specifications we presented above is called *verbose*. In a *verbose* specification, the components $\mathcal{A}$ and P are supposed to be universally quantified and that means these components could be thought as global data structures through the definition of the specification.

There is another form of flow logic specifications which records information about a process locally. Its judgements are in the form of

$$\mathcal{A} \models P : \mathcal{A}' \ \textbf{iff} \ \text{ some logic formula holds}$$

where $\mathcal{A}'$ contains the analysis result about the behavior of the process P only. In Flow Logic this form of specifications is called *succinct*. The succinct components in the left hand side of **iff** are considered to be existentially quantified so that their scope is constrained in the left part of the clause and not known to the entire analysis.

A verbose flow logic specification is more implementation oriented while its succinct version sometimes is more elegant. One can always convert a succinct flow logic specification into its equivalent verbose form with the technique developed in [14].

In next section we shall present the control flow analysis of ABoxed Ambients as well as illustrate the concepts in Flow logic discussed so far.

3.2 A Control Flow Analysis for ABoxed Ambients

ABoxed Ambients is an ambient calculus designed to model security issues in hierarchical network applications, in particular, cryptographic protocols. The control flow analysis presented in this section aims at approximating the communication of different locations as well as validating the authenticity of a protocol based on the information of communication analysis [3]. In reference monitor setting (defined in Chapter 2) we need to *safely* approximate when the reference monitor may abort the computation of a process *P*.

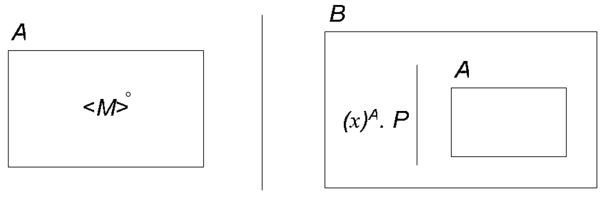
3.2.1 Domain of the Control Flow Analysis

To describe the communication in hierarchical locations, we need at least an analysis component κ to record the tuples that may appear in the mailboxes of locations. Additionally we also need another two analysis components: (1) An analysis component γ that tracks the contents of an ambient including ambients and capabilities, and (2) an analysis component ρ that records the tuples a variable may be bound to.

Borrowing the idea from LySa the analysis only records the canonical representatives of names and variables of a process. There are at least two reasons for adopting canonical values instead of semantic values: The first is to deal with arbitrary naming space; the second is to ensure the analysis result meaningful under α – conversion. We explain them one by one as below.

Even though a process can use replication together with restriction to create unlimited number of names at run-time, our analysis simply treats them as the same one.

As the result two names distinguished in the semantics are not distinguishable in the analysis components if they share the same canonical name. This ambiguity can cause the imprecision of analysis result as illustrated by the following example process:



As figure shows there are two ambients named *A* and semantically they are two different ambients. According to the semantics presented in Chapter 2 the input of the ambient *B* can not be executed and therefore the variable *x* has no bound value. The analysis, nevertheless, will combine the mailbox of the two ambients having the same name A and attempt to allow that the input primitive can be executed successfully. As the result, the canonical variable $\lfloor x \rfloor$ may be bound to the canonical name $\lfloor M \rfloor$.

However this imprecision in the analysis only causes the analysis to over approximate the behavior of a process. Thus we still safely describe the property of a process as well as not lose any necessary information.

Another benefit of using the canonical name is to remain the control flow estimate correct for the $\alpha-$converted system. This can be illustrated by considering the following example process

$$(\nu\ A)(\nu\ packet)A[packet[0]]$$

Suppose our analysis can correctly estimate that $\lfloor packet \rfloor$ occurs inside $\lfloor A \rfloor$ but not vice versa. We consider to $\alpha-$convert A to A' and $packet$ to $packet'$ respectively and the above process is changed to

$$(\nu\ A')(\nu\ packet')A'[packet'[0]]$$

Since $\alpha-$conversion is stable under the canonical name, we are sure that $\lfloor A \rfloor = \lfloor A' \rfloor$ and $\lfloor packet \rfloor = \lfloor packet' \rfloor$. As the result our estimate is still correct under the $\alpha-$conversions.

Actually a very similar concept can be found at [3] in which every name is given a type called group. This group is stable under $\alpha-$conversion and the analysis only distinguishes between two groups.

The formal definitions of the four analysis components are given as below:

(1) A component keeps track of the contents of the mailboxes:

$$\kappa: \lfloor \mathcal{C} \rfloor \to \mathcal{P}((\lfloor \mathcal{C} \rfloor \cup \lfloor M \rfloor)^*)$$

that for every ambient name records the tuples of messages that may show up in an ambient's mailbox.

(2) A component records the contents of given ambients

$$\gamma: \lfloor \mathcal{C} \rfloor \to \mathcal{P}(\lfloor \mathcal{C} \rfloor \cup \lfloor M \rfloor)$$

that for every ambient name (canonical version) approximates which ambients and capabilities may be contained

(3) A component keeps track of the relevant bindings of names:

$$\rho: \lfloor \mathcal{X} \rfloor \to \mathcal{P}((\lfloor \mathcal{C} \rfloor \cup \lfloor M \rfloor)^*)$$

that for every variable records the tuples of possible values including names and capabilities.

(4) A component describes the possible violation of authenticity:

$$\varphi:\ \mathcal{P}(\mathcal{D}\times\mathcal{D})$$

Since we can not restrict any variable, there are always limited variables introduced in a process. Thus it seems that the canonical version for variables is introduced for only the second reason discussed above. But soon this concept of canonical representative will also be useful to handle the case of unlimited variables the attacker may occupy.

3.2.2 Definition of the Control Flow Analysis

The judgement of the analysis takes the form

$$(\gamma,\kappa,\rho)\models^{\mu} P:\varphi$$

expressing that when P is contained by an ambient having canonical name μ ; (γ,κ,ρ) is an acceptable analysis estimate for the process P – meaning that γ includes all ambients and capabilities that may appear inside the ambients as P evolves, κ reflects all messages may be available in the mailboxes of the ambient μ and ρ will contain all the name bindings that occur. Here φ may be an empty set or it may report errors of the form (ℓ,ℓ') which tell us some message encrypted at ℓ was unexpectedly decrypted at ℓ'.

Analysis Specification:

Table 3-1 (on the next page) specifies the analysis of composite process (non-communication rules). These rules require that each acceptable analysis (γ,κ,ρ) of a composite process is also a valid analysis estimation of its sub-process.

$(\gamma,\kappa,\rho) \models^{*} (\nu\ n)P : \varphi$	iff $(\gamma,\kappa,\rho) \models^{*} P : \varphi$
$(\gamma,\kappa,\rho) \models^{*} (\nu k\ n)P : \varphi$	iff $(\gamma,\kappa,\rho) \models^{*} P : \varphi$
$(\gamma,\kappa,\rho) \models^{*} 0 : \varphi$	iff true
$(\gamma,\kappa,\rho) \models^{*} P_1 \mid P_2 : \varphi$	iff $(\gamma,\kappa,\rho) \models^{*} P_1 : \varphi \wedge (\gamma,\kappa,\rho) \models^{*} P_2 : \varphi$
$(\gamma,\kappa,\rho) \models^{*} !P : \varphi$	iff $(\gamma,\kappa,\rho) \models^{*} P : \varphi$
$(\gamma,\kappa,\rho) \models^{*} N[P] : \varphi$	iff $\forall \mu \in \mathcal{N}_\rho(N) : \mu \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\mu} P : \varphi$
$(\gamma,\kappa,\rho) \models^{*} \text{in}\ N.P : \varphi$	iff $\mathcal{M}_\rho(\text{in}\ N) \subseteq \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{*} P : \varphi \wedge$ $\forall \text{in}\ \mu \in \mathcal{M}_\rho(\text{in}\ N) : \varphi_{\text{in}}(\mu)$
$(\gamma,\kappa,\rho) \models^{*} \text{out}\ N.P : \varphi$	iff $\mathcal{M}_\rho(\text{out}\ N) \subseteq \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{*} P : \varphi \wedge$ $\forall \text{out}\ \mu \in \mathcal{M}(\text{out}\ N) : \varphi_{\text{out}}(\mu)$
$(\gamma,\kappa,\rho) \models^{*} N.P : \varphi$	iff $\mathcal{M}_\rho(N) \cap M \subseteq \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{*} P : \varphi \wedge$ $\forall \text{in}\ \mu \in \mathcal{M}_\rho(N) : \varphi_{\text{in}}(\mu) \wedge$ $\forall \text{out}\ \mu \in \mathcal{M}_\rho(N) : \varphi_{\text{out}}(\mu)$

Table 3-1: **Analysis specification (Part 1):** $(\gamma,\kappa,\rho) \models^{\mu} P : \varphi$

In the first two clauses the analysis ignores the two kinds of restrictions. The clause for parallel construct requires that the analysis result is also valid for both the two branches. The rule for replication ignores the multiplicity and continues checking the continuation process *P*. That means the multiplicity does not add any new constraint into the analysis.

The clause for ambients makes sure that the analysis estimate γ records that the canonical representative μ of the naming *N* is inside the current ambient called * and then the sub-process *P* should be analyzed under the updated ambient μ. Since the naming *N* could also be given by a variable, we use the function $\mathcal{N}_\rho : \mathcal{V} \rightarrow \mathcal{P}(\lfloor \mathcal{C} \rfloor)$ to map namings to their canonical values. It is specified in the upper part of Table 3-2 below.

$$\mathcal{N}_\rho(x) = \rho(\lfloor x \rfloor) \cap \lfloor \mathcal{C} \rfloor$$
$$\mathcal{N}_\rho(n) = \{\lfloor n \rfloor\}$$

$$\mathcal{M}_\rho(\text{in}\ N) = \{\text{in}\ \mu \mid \mu \in \mathcal{N}_\rho(N)\}$$
$$\mathcal{M}_\rho(\text{out}\ N) = \{\text{in}\ \mu \mid \mu \in \mathcal{N}_\rho(N)\}$$
$$\mathcal{M}_\rho(x) = \rho(\lfloor x \rfloor)$$
$$\mathcal{M}_\rho(n) = \{\lfloor n \rfloor\}$$

Table 3-2: **Functions** $\mathcal{N}_\rho$ **and** $\mathcal{M}_\rho$

As the table shows, capabilities are excluded from the domain as well as range of the function $\mathcal{N}_\rho$. Thus to handle in-capability and out-capability we need one more function $\mathcal{M}_\rho : M \to \mathcal{P}(\lfloor M \rfloor)$ that map capabilities to sets of canonical capabilities.

The last three clauses deal with prefixed processes and are explained below:

(1) **In-capability**. It first records the set of the actual capabilities and analyzes the continuation process. The "closure condition" φ_{in} (See the upper part of the Table 3-3) reflects the semantics of in-capability into the analysis. The precondition of the universally quantified implication specifies that if μ^a has the capability in μ, μ^p is the enclosing ambience of μ^a and μ^a has a sibling μ, then the analysis records that μ^a may be inside μ.

(2) **Out-capability**. The clause "closure condition" φ_{out} (See lower part of the Table 3-3) is quite similar to φ_{in} except that now it corresponds to the operational semantics for the out-capability.

(3) **Naming *N***. Here *N* can be a name or a variable. If it is a name or a variable bound with a name, there is not corresponding semantic rule for the process *N.P*. Thus the analysis omits *N* and simply analyzes the continuation process. Otherwise the closure conditions φ_{in} or φ_{out} will be applied for the potential capabilities of the value of *N*.

$$
\begin{array}{ll}
\varphi_{\text{in}}(\mu) & \text{iff } \forall \mu^a, \mu^p : \text{in } \mu \in \gamma(\mu^a) \wedge \\
& \qquad \mu^a \in \lfloor \mathcal{C}_P \rfloor \wedge \\
& \qquad \mu^a \in \gamma(\mu^p) \wedge \\
& \qquad \mu \in \gamma(\mu^p) \\
& \qquad \Rightarrow \mu^a \in \gamma(\mu)
\end{array}
$$

$$
\begin{array}{ll}
\varphi_{\text{out}}(\mu) & \text{iff } \forall \mu^a, \mu^p : \text{out } \mu \in \gamma(\mu^a) \wedge \\
& \qquad \mu^a \in \lfloor \mathcal{C}_P \rfloor \wedge \\
& \qquad \mu^a \in \gamma(\mu) \wedge \\
& \qquad \mu \in \gamma(\mu^g) \\
& \qquad \Rightarrow \mu^a \in \gamma(\mu^g)
\end{array}
$$

Table 3-3: **Closure conditions $\varphi_{\text{in}}(\mu)$ and $\varphi_{\text{out}}(\mu)$**

Example 3.1 We consider the process

$$A[\,p[\text{out } A.\text{in } B]\,] \mid B[\,]$$

of example 2.1 again. The analysis estimate

$$\begin{array}{ll} \gamma(n_\bigstar) = \{A, B, p\} & \kappa(n_\bigstar) = \kappa(A) = \kappa(B) = \kappa(p) = \varnothing \\ \gamma(A) = \{p\} & \varphi = \varnothing \\ \gamma(p) = \{\text{out}\, A, \text{in}\, B\} & \\ \gamma(B) = \{p\} & \end{array}$$

will show the possible behavior of the process. It can be verified that this estimate is the best estimate found by the analysis specified in Table 3-1. According to the estimate γ, *p* may turn up inside $n_\bigstar$, *A* and *B*. As there is no communication in the process, κ is empty for every ambient. So is φ.

Note that the name recorded by the analysis is their canonical representative and thus the estimate is stable under $\alpha-$conversion.

□

In Table 3-4 (on the next page) we define the clauses of inputs and outputs for each of the three directions. Their explanation is as below.

For the local output, the clause (1) collects the possible values of each capability M_i (2) puts all k-tuples of messages of the form $< v_1, v_2, ..., v_k >_\ell$ [dest $\mathcal{L}$] taken from $\mathcal{M}(M_1) \times \mathcal{M}(M_2) \times ... \times \mathcal{M}(M_k)$ into the local mailbox $\kappa(*)$.

For any available k-tuple of messages $<v_1, \cdots, v_k>_\ell$ [dest $\mathcal{L}$] in local mailbox $\kappa(*)$, the clause for local input (1) tests whether the first j elements of $<v_1, \cdots, v_j, v_{j+1}, ..., v_k>$ are pointwise included in $\mathcal{M}(M_i)$. In case the check is successful, then (2) the values $v_{j+1}, ..., v_k$ are included into the analysis estimates for the variables $x_{j+1}, \cdots, x_k$, respectively. At last, (3) the $\varphi-$component must include (ℓ, ε) if the destination assertion might be violated, i.e. if $\varepsilon \notin \mathcal{L}$. Here a special crypto-point ε is added by the analysis and reserved for annotating any non-encrypted message. Therefore if $\varepsilon \notin \mathcal{L}$ then the candidate message is not supposed to be received by local input but by child input as only child input can have crypto-point other than ε. The origin assertion $\mathcal{D}$, also added automatically in the analysis, means that local input does not care where the message comes from.

In the clause for the communication with a child, we (1) collect all the possible values μ of the ambient *N* and (2) check whether the corresponding ambient might show up in the ambient *. For each successful value μ, the communication is recorded by little adjusting the clauses for local communication on the annotation: the messages of output have the form $< v_1, v_2, ..., v_k >_\varepsilon$ [dest $\mathcal{D}$] where the crypto-point ε and the default destination $\mathcal{D}$ are added by the analysis silently for the judgement of $\varphi-$component. Since the input the cryto-point ℓ' and the origin assertion $\mathcal{L}'$ have been declared explicitly which the analysis need not insert or modify. As the result, the $\varphi-$component should include (ℓ, ℓ') if the destination **or** origin assertions are violated, i.e. $(\ell \in \mathcal{L}')$ or $(\ell' \in \mathcal{L})$.

$$
\begin{array}{ll}
(\gamma,\kappa,\rho) \models^{*} < M_1,\cdots,M_k >_{\ell}^{\circ} [\text{dest } \mathcal{L}]{:}\varphi & \text{iff } \forall v_1,\cdots,v_k{:}\ \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\rho}(M_i) \\
 & \quad \Rightarrow <v_1,\cdots,v_k>_{\ell} [\text{dest } \mathcal{L}] \in \kappa(*) \\
(\gamma,\kappa,\rho) \models^{*} (M_1,\cdots,M_j; x_{j+1},\cdots,x_k)^{\circ}.P{:}\varphi & \text{iff } <v_1,\cdots,v_k>_{\ell} [\text{dest } \mathcal{L}] \in \kappa(*){:} \\
 & \quad \wedge_{i=1}^{j} v_i \in \mathcal{M}_{\rho}(M_i) \\
 & \quad \Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(x_i) \wedge \\
 & \quad (\neg \text{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L}) \Rightarrow (\ell,\varepsilon) \in \varphi) \wedge \\
 & \quad (\gamma,\kappa,\rho) \models^{*} P{:}\varphi \\
(\gamma,\kappa,\rho) \models^{*} < M_1,\cdots,M_k >^{N} {:}\varphi & \text{iff } \forall \mu \in \mathcal{N}_{\rho}(N) : \mu \in \gamma(*) \wedge \\
 & \quad \forall v_1,\cdots,v_k{:}\ \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\rho}(M_i) \\
 & \quad \Rightarrow <v_1,\cdots,v_k>_{\varepsilon} [\text{dest } \mathcal{D}] \subseteq \kappa(\mu) \\
(\gamma,\kappa,\rho) \models^{*} (M_1,\cdots,M_j; x_{j+1},\cdots,x_k)_{\ell'}^{N} [\text{orig } \mathcal{L}'].P{:}\varphi & \text{iff } \forall \mu \in \mathcal{N}_{\rho}(N) : \mu \in \gamma(*) \wedge \\
 & \quad \forall < v_1,\cdots,v_k >_{\ell} [\text{dest } \mathcal{L}] \in \kappa(\mu){:} \\
 & \quad \wedge_{i=1}^{j} v_i \in \mathcal{M}_{\rho}(M_i) \\
 & \quad \Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(x_i) \wedge \\
 & \quad (\neg \text{RM}(\ell,\mathcal{L}',\ell',\mathcal{L}) \Rightarrow (\ell,\ell') \in \varphi) \wedge \\
 & \quad (\gamma,\kappa,\rho) \models^{*} P{:}\varphi \\
(\gamma,\kappa,\rho) \models^{*} < M_1,\cdots,M_k >^{\uparrow} {:}\varphi & \text{iff } \forall \mu : * \in \gamma(\mu) \wedge \\
 & \quad \forall v_1,\cdots,v_k{:}\ \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\rho}(M_i) \\
 & \quad \Rightarrow <v_1,\cdots,v_k>_{\varepsilon} [\text{dest } \mathcal{D}] \subseteq \kappa(\mu) \\
(\gamma,\kappa,\rho) \models^{*} (M_1,\cdots,M_j; x_{j+1},\cdots,x_k)^{\uparrow}.P{:}\varphi & \text{iff } \forall \mu : * \in \gamma(\mu) \wedge \\
 & \quad \forall < v_1,\cdots,v_k >_{\ell} [\text{dest } \mathcal{L}] \in \kappa(\mu)){:} \\
 & \quad \wedge_{i=1}^{j} v_i \in \mathcal{M}_{\rho}(M_i) \\
 & \quad \Rightarrow v_{j+1} \in \rho(x_j) \wedge \cdots \wedge v_k \in \rho(x_k) \wedge \\
 & \quad (\neg \text{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L}) \Rightarrow (\ell,\varepsilon) \in \varphi) \wedge \\
 & \quad (\gamma,\kappa,\rho) \models^{*} P{:}\varphi
\end{array}
$$

Table 3-4: Analysis specification (Part 2): $(\gamma,\kappa,\rho) \models^{\mu} P : \varphi$

The clause for the communication with an enclosing ambient (1) obtains all the possible ambients μ that might be parent of the ambient * and (2) for each possible ambient μ the communication is recorded by adopting the clause for local communication except that the crypto-point ε and the default destination $\mathcal{D}$ are inserted or used by both input and output as there is not any annotation for the two communication primitives.

A little imprecise of the analysis is that the massage sending out to a child or parent is always successfully delivered to the mailbox of the ambients even though there may be not a corresponding input to receive message. This can be illustrated by following example with the annotation removed,

$$(;x)^{a} \mid a[b[<m>^{\uparrow}]]$$

The communication will not succeed according to the semantics, formally

$$(;x)^{a} \mid a[b[<m>^{\uparrow}]] \not\rightarrow .$$

But the analysis will pretend it succeeds. In fact, what the analysis approximates is the program (for the presentation clarity we omit the annotation here)

$$(;x)^{a} \mid a[(y)^{\circ}.<y>^{\circ} \mid b[<m>^{\uparrow}]]$$

Another limitation of our analysis is that the analysis is flow-insensitive. That is we can record the presence of capabilities but not the order in which they appear. Thus we can not capture the order in which capabilities are executed. Also we can not capture the order of the presence of inputs. To record the order of capabilities, one may refer to [26].

Example 3.2 We consider the process

$$\begin{aligned}&A[p[\text{out } A.\text{in } B.(<m_1>^{\uparrow} \mid <m_2>^{\circ}_{A} \text{ [dest B]})]] \mid \\ &B[(;x_1)^{\circ}.P \mid (;x_2)^{p}_{B}[\text{orig A}].P']\end{aligned}$$

presented in Example 2-3. The analysis estimate

$$\begin{array}{llll}\gamma(n_\star)=\{A,B,p\} & \kappa(n_\star)=\{m_1\} & \rho(x_1)=\{m_1\} & \varphi=\emptyset \\ \gamma(A)=\{p\} & \kappa(A)=\{m_1\} & \rho(x_2)=\{m_2\} & \\ \gamma(p)=\{\text{out } A, \text{ in } B\} & \kappa(B)=\{m_1\} & & \\ \gamma(B)=\{p\} & \kappa(p)=\{m_2\} & & \end{array}$$

will show the possible behavior of the process. We can check the estimate with the rules in Table 3-1 and 3-4 and it can be verified that this estimate is the best estimate found by the analysis. The estimate component φ is empty which implies authenticity is guarantied.

□

3.3 Correctness of the Analysis

We need prove the correctness of our analysis with respect to the operational semantics defined in Table 2-5 and 2-6. As we explained in sub-section 3.1.1, this amounts to proving a subject reduction result for both the standard and the reference monitor semantics: If $(\gamma,\kappa,\rho) \models^{\mu} P:\varphi$, then the same estimate is also valid for all the states that may be passed through in the execution of *P*.

For the convenience of the proof, we first prove two lemmas.

Lemma 1: Substitution

$(\gamma,\kappa,\rho) \models^{\mu} P:\varphi$ and $\lfloor M' \rfloor \in \rho(\lfloor x \rfloor)$ imply $(\gamma,\kappa,\rho) \models^{\mu} P\{x \leftarrow M'\}:\varphi$.

The proof is done by straightforward induction on process *P* and applying induction hypothesis on any subprocess. The proof relies on the fact the $\alpha-$conversion is stable under the canonical name.

Lemma 2: Congruence

If $P \equiv Q$ then $(\gamma,\kappa,\rho) \models^{\mu} P:\varphi$ *iff* $(\gamma,\kappa,\rho) \models^{\mu} Q:\varphi$.

The proof is by induction in the definition of $P \equiv P'$ defined in Table 2-2. The most interesting case is when the considered process is $!P$, we need proof that $(\gamma,\kappa,\rho) \models^{\mu} P\,|\,!P:\varphi$. This can be justified by below calculation since by the definition of the analysis:

$$\begin{aligned}(\gamma,\kappa,\rho) \models^{\mu} !P:\varphi \;\; &\text{iff } (\gamma,\kappa,\rho) \models^{\mu} P:\varphi \\ &\text{iff } (\gamma,\kappa,\rho) \models^{\mu} P:\varphi \wedge (\gamma,\kappa,\rho) \models^{\mu} P:\varphi \\ &\text{iff } (\gamma,\kappa,\rho) \models^{\mu} P:\varphi \wedge (\gamma,\kappa,\rho) \models^{\mu} !P:\varphi \\ &\text{iff } (\gamma,\kappa,\rho) \models^{\mu} P\,|\,!P:\varphi\end{aligned}$$

Theorem 1: Subject reduction

If $P \rightarrow_{\mathcal{R}} Q$ and $(\gamma,\kappa,\rho) \models^{\mu} P:\varphi$ then $(\gamma,\kappa,\rho) \models^{\mu} Q:\varphi$.

The formal proof is given in Appendix D.1, for your reference.

Last we declare that

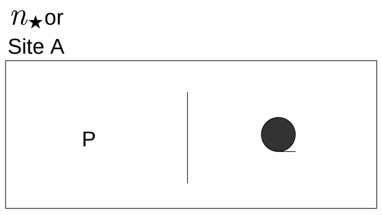
Theorem 2: Static check for reference monitor

If $(\gamma,\kappa,\rho) \models^{\mu} P:\varnothing$ then RM can not abort *P*.

It shows that the analysis can correctly estimate when we can dispense with the reference monitor safely. To prove the theorem, we show there is no Q, Q' such that $P \rightarrow^{*} Q \rightarrow Q'$ and $P \rightarrow^{*}_{\mathsf{RM}} Q \not\rightarrow_{\mathsf{RM}}$. This is proved by contradiction: Suppose such Q, Q' exist, then by Theorem 1 we have that $P \rightarrow^{*} Q$ gives $(\gamma,\kappa,\rho) \models^{\mu} P:\varnothing$. Since by the general subject reduction (proved in Appendix D.1), $Q \rightarrow Q'$ gives $Q \rightarrow_{\mathsf{RM}} Q'$ that is a contradiction.

4 Modeling the Attacker

There may be malicious attackers in a network environment where cryptographic protocols are applied. To validate a protocol, we need model the behaviors of attackers and let attackers and protocols execute in parallel. Formally we write P_{sys} for the implementation of the protocol. The attacker shall be declared as one of top level processes in the distinguished ambient $n_\star$ or any site ambients of $P_{sys.}$ This can be graphically described as below



There is no default location for the attacker and where they appear depends on our assumption. In this thesis the process which have no attacker inside, i.e. P_{sys} , is called *target process*.

With ambient calculi, there may be several different kinds of attackers that one can model. The basic capabilities of attackers follow the classical Dolev-Yao condition [7] under which attackers can

(1) receive messages by eavesdropping,

(2) decrypt messages using the key they knows,

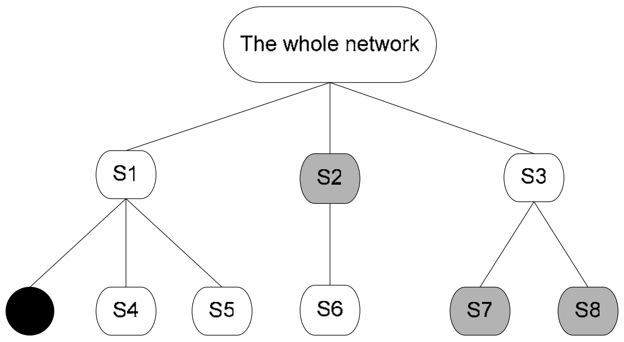
(3) construct new messages (encrypted or plain),

(4) send messages they have

Subject to the hierarchy of the network environment, the conditions (1) and (4) become unclear and need further clarifying. For the first condition we need declare the communication of which location an attacker can overhear as there may be many locations that an attacker can nest in and also communication happens in several different locations. For the same reason, we should also make it clear in the fourth condition that which location(s) an attacker can deliver message to.

Since our protocols run in hierarchical networks in which communication inside a local network and its enclosed network(s) are thought to be secret to outside, we here suggest attackers can only eavesdrop on the communication of their nested places.

For the fourth condition, we suggest allowing attackers to deliver messages to every site, called *A*, that are *reachable* from the attacker-nested sites, called $S_\bullet$; here *reachable* means that there is a route (consists of a series of sites) from $S_\bullet$ to *A* along which every name of the site is known by the attacker. We illustrate the concept by the following example. Suppose there is a network whose structure is represented by a tree as below



where the bullet circle denotes the attacker, white color circles denote the sites whose names are known by the attacker and gray color circles are nodes unknown to the attacker. The root of the tree, the whole network, is always known by attackers. Then the reachable sites are S1, S3, S4 and S5. But S6 is not reachable as the name of S2 is not available to attackers.

Considering the multiple locations attackers may nest in, we need one more condition to specify the relation between attackers. We shall suppose they share all the information and knowledge each other via a private channel. By this way attackers maximize their capabilities in attacking a network.

Summarize the above discussion, the adjusted Dolev-Yao condition with respect to the hierarchical network is written as below.

(1) Attackers can eavesdrop on only the message presenting in their locations;

(2) They can decrypt message using the key he knows;

(3) They can construct new messages (encrypted or plain);

(4) They can send messages they have to their reachable sites;

(5) Initially they have some knowledge about the environment; a private channel is used by attackers to share knowledge and information with each other.

The characteristic of this kind of attackers is that they have fixed work locations given by our assumption. Same with the Dolev-Yao attacker, the attacker only works on the network but does not intrude into the participants of the protocols. That is the packets they send are just data and not executive. For our calculus specifically "executive" means a packet can not collect information from the enclosing site by the input.

With ambient calculus, we can further extend the attacker's capabilities in three ways at least. First an attacker can also be equipped with mobility and therefore can move around any their reachable sites. The capability of the attackers may be too powerful. Under most cases of protocol validation, however, we still prefer to be clear that where an attacker can access to.

The second way of extending the capabilities of attackers is to allow the attacker's packet to collect information of the sites which the packet passes by, and send the information back to the attacker; that is attacker can compose packets including the input. This kind of attackers could be modeled by ambient calculi but what is more this extension benefits protocol validation needs further investigation. Moreover, in the real life, any code can only be executed in some environment, i.e. computers, servers, etc. But the calculus would also allow the execution of an input in a local network. Thus the attacker can acquire more information in our model than he is expected to get in a real network.

The third extension to the attacker capabilities allows attackers to be attached in a packet and be sent into other sites along with the packet. Again this kind of attackers acquires the capability to attack any their reachable sites. Similar to the attacker with mobility, they are somewhat too strong for protocol validation. But they may be useful for analyzing the behaviors of some virus i.e. Troy house. If that is the case, we need some mechanism to model the activation of the malicious code inside the packet.

In this thesis we shall adopt and implement the adjusted Dolev-Yao condition to capture the capabilities of attackers. Furthermore we claim that the processes of attacker should follow the rules of the well-formednesses $\Gamma \vdash wf_{\mathrm{S}}(P_\star)$ and $\Gamma \vdash wf_{\mathrm{P}}(P_\star)$ and call such attackers well-formed. Inspired by the studies of LySa, we aim at finding a formula $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ which characterizing all well-formed attackers. Accordingly whenever an estimate $(\gamma,\kappa,\rho,\varphi)$ satisfies $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ then we are sure $(\gamma,\kappa,\rho) \vDash^{\mu} Q : \varphi$ for all well-formed attackers Q. The technique used here, therefore, is quite similar to that used in LySa [2].

To benefit the control flow analysis we shall say that a target process P is of type $(\mathcal{N}_{\mathrm{f}}, \mathcal{A}_{\mathcal{K}})$ whenever: (1) P is closed, (2) it is a well-formed site, i.e. $\Gamma \vdash wf_{\mathrm{S}}(P)$, (3) its free name are in $\mathcal{N}_{\mathrm{f}}$, (4) all the arities used for input and output are in $\mathcal{A}_{\mathcal{K}}$. We can easily find minimal $\mathcal{N}_{\mathrm{f}}$ and $\mathcal{A}_{\mathcal{K}}$ such that P is of type $(\mathcal{N}_{\mathrm{f}}, \mathcal{A}_{\mathcal{K}})$ and it is also of type $(\mathcal{N}'_{\mathrm{f}}, \mathcal{A}'_{\mathcal{K}})$ given that $\mathcal{N}_{\mathrm{f}} \subseteq \mathcal{N}'_{\mathrm{f}}$ and $\mathcal{A}_{\mathcal{K}} \subseteq \mathcal{A}'_{\mathcal{K}}$.

We claim the ability of the well-formed attacker process Q to use:

- Additional free names may be masked by restricting the names so as to become local within Q,
- A "private channel" based on k-ary communication for $k \notin \mathcal{A}_{\mathcal{K}}$ does not increase its computational power.

To have control over the canonical names and variables attackers may use, we first inspect a target process to find the finite set $\mathcal{N}_{\mathrm{c}}$ of all canonical names used in the target process and the finite set $\mathcal{X}_{\mathrm{c}}$ of all canonical variables used. Then we postulate a fresh canonical names $n_\bullet$ not in $\mathcal{N}_{\mathrm{c}}$ and a fresh canonical variable $z_\bullet$ not in $\mathcal{X}_{\mathrm{c}}$. Provided there is a process Q of type $(\mathcal{N}_{\mathrm{f}}, \mathcal{A}_{\mathcal{K}})$, we can translate it into the semantically equivalent process $\overline{Q'}$ by the following steps: (1) all restrictions $(\nu\ n)P$ or $(\nu k\ n)P$ are $\alpha-$converted into $(\nu\ n')P'$ or $(\nu k\ n')P'$ in which n' has the canonical name $n_\bullet$; (2) all occurrences of variables x_i in inputs $(M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\eta}.P$ are $\alpha-$converted into the variables x'_i whose canonical name is $z_\bullet$. By this way, all the canonical names and variables that attackers may have are coalesced into $n_\bullet$ and $z_\bullet$ and therefore only finitely many canonical names and variables are used in the process $\overline{Q'}$.

For the annotations of attacker-encrypting messages, we use the trivial ones i.e. [dest $\mathcal{D}$] or [orig $\mathcal{D}$] for authentication intentions of attackers and all crypto-points of attackers are coalesced into one crypto-point $\ell_\bullet$ which is fresh to the target process *P*. The resulting process is written to be $\overline{Q}$ (here the annotations of attackers are not used to express the intentions of a protocol but to comply with the syntax.).

We then define the formula $\mathcal{F}_{\text{RM}}^{\text{A\_DY}}$ of type $(\mathcal{N}_{\text{f}}, \mathcal{A}_{\mathcal{K}})$ for expressing the adjusted Dolev-Yao condition for ABoxed Ambients. The formula is defined as the conjunction of the five components (may have sub-components) in Table 4-1 where $*$ represents the location of the attacker.

Component 1:

1.1 $\wedge_{k \in A_k} \ \forall < v_1,...,v_k >_\varepsilon \text{[dest } \mathcal{D}] \in \kappa(*)\text{: } \wedge_{i=1}^{k} \text{genCap}(v_i)$

1.2 $\wedge_{k \in A_k} \ \forall \mu : \mu \in \gamma(*) \wedge \mu \in \mathcal{C}_p \wedge$

$<v_1,...,v_k >_\ell \text{[dest } \mathcal{L}] \in \kappa(\mu) \Rightarrow < \mu, v_1,...,v_k >_\ell \text{[dest } \mathcal{L}] \in \rho(t_\bullet)$

Component 2:

2 $\wedge_{k \in A_k} \ \forall < \mu, v_1,...,v_k >_\ell \text{[dest } \mathcal{L}] \in \rho(t_\bullet) : \mu \in \rho(z_\bullet) \Rightarrow$

$(\wedge_{i=1}^{k} v_i \in \rho(z_\bullet) \wedge (\neg \text{RM}(\ell, \mathcal{C}, \ell_\bullet, \mathcal{L}) \Rightarrow (\ell, \ell_\bullet) \in \varphi))$

Component 3:

3.1 $\wedge_{k \in A_k} \ \forall v_1,...,v_k : \wedge_{i=1}^{k} v_i \in \rho(z_\bullet)$

$\forall \mu \in \rho(z_\bullet) : \mu \neq in(m) \ \wedge \ \mu \neq out(m) \wedge \mu \in \mathcal{C}_p$

$\Rightarrow \ < \mu, v_1,...,v_k >_{\ell_\bullet} \text{[dest } \mathcal{D}] \in \rho(t_\bullet)$

3.2 $\wedge_{k \in A_k} \ \forall v_1,...,v_k : \wedge_{i=1}^{k} v_i \in \rho(z_\bullet)$

$\forall \mu\text{: } \mu \in \text{Reach}(*)$

$\Rightarrow \ < \mu, v_1,...,v_k >_\varepsilon \text{[dest } \mathcal{D}] \in \rho(t_\bullet)$

Component 4:

4.1 $\forall t, m\text{: } t \in \rho(z_\bullet) \ \wedge \ t \neq \text{in}(m) \ \wedge \ t \neq \text{out}(m) \wedge t \in \mathcal{C}_p \wedge$

$\forall \mu\text{: } \mu \in \text{Reach}(*)$

$\Rightarrow t \in \gamma(\mu)$

4.2 $\wedge_{k \in A_k} \ \forall < \mu, v_1,...,v_k >_\ell \text{[dest } \mathcal{L}] \in \rho(t_\bullet)\text{:} < v_1,...,v_k >_\ell \text{[dest } \mathcal{L}] \in \kappa(\mu)$

Component 5:

5.1 $\{n_\bullet\} \cup \mathcal{N}_{\text{f}} \cup \mathcal{N}_{\text{v}} \ \cup \{n_\star\} \subseteq \rho(z_\bullet)$

5.2 $\forall m \in \{n_\bullet\} \cup \mathcal{N}_{\text{f}} \cup \mathcal{N}_{\text{v}} \ : \ \text{in}(m) \in \rho(z_\bullet) \wedge \text{out}(m) \in \rho(z_\bullet)$

Table 4-1: Adjusted Dolev-Yao condition.

As the table shows, we use an additional fresh variable $t_{\bullet}$ to record the messages attackers may have. This should have no semantic consequence but for the convenience of presenting the adjusted Dolev-Yao condition. Each component in Table 4-1 corresponds to an item of the adjusted Dolev-Yao condition and we explain every component below.

The attacker initially has some knowledge about the environment and this is represented by the fifth component including two sub-components 5.1 and 5.2. Here $\mathcal{N}_{\mathrm{v}}$ is the set of all the restricted names, the ranges of which cover the process of the attacker excepting for the secret restricted name introduced by ($\nu k\ n$). Note that the attacker may be located in several different places, $\mathcal{N}_{\mathrm{v}}$ is the set including all restricted names whose range cover one or several attackers. Furthermore once the attacker knows a name *n*, he also knows the corresponding capabilities in(*n*) and out(*n*). As we suppose no ambient move out of the *background envieronment* formalized as the ambient $n_{\bigstar}$, the attacker should not have capabilities in($n_{\bigstar}$) or out($n_{\bigstar}$) either.

The first component expresses that the attacker can eavesdrop on messages presenting in his location. Especially, the sub-component 1.2 tell us the formula only records the message in the packet that may show up in the ambient $*$. The function genCap:$\lfloor \mathrm{Cap} \rfloor \to \mathcal{P}(z_{\bullet} \to \lfloor \mathrm{Cap} \rfloor)$ is given by the conjunction of the three components

$$(1)\ \forall t: v_i = \mathrm{in}(t) \Rightarrow \mathrm{out}(t) \in \rho(z_{\bullet}) \wedge t \in \rho(z_{\bullet})$$
$$(2)\ \forall t: v_i = \mathrm{out}(t) \Rightarrow \mathrm{in}(t) \in \rho(z_{\bullet}) \wedge t \in \rho(z_{\bullet})$$
$$(3)\ \forall t: v_i \neq \mathrm{in}(t) \wedge v_i \neq \mathrm{out}(t) \Rightarrow \mathrm{out}(v_i) \in \rho(z_{\bullet}) \wedge \mathrm{in}(v_i) \in \rho(z_{\bullet})$$

The second component expresses that the attacker decrypts messages with the key already known. For the third component, the two sub-components represent that the attacker constructs new encryptions (3.1) and plain-text messages (3.2) recorded by the variable $t_{\bullet}$. The function Reach: $\lfloor \mathcal{C}_s \rfloor \to \mathcal{P}(\lfloor \mathcal{C}_s \rfloor)$ is given by the conjunction of the following components

$$(1)\ \forall \mu \in \mathrm{SITE}\text{: } \mu \in \mathrm{Reach}(\mu)$$
$$\begin{aligned}(2)\ & \forall \mu_1 \in \mathrm{SITE},\ \forall \mu_2 \in \mathrm{SITE}\text{:}\\ & \quad \mu_1 \in \mathcal{R}(z_{\bullet}) \wedge \mu_2 \in \mathcal{R}(z_{\bullet}) \wedge \mu_2 \in \gamma(\mu_1)\\ & \quad \Rightarrow \mu_2 \in \mathrm{Reach}(\mu_1)\end{aligned}$$
$$\begin{aligned}(3)\ & \forall \mu_1 \in \mathrm{SITE},\ \forall \mu_2 \in \mathrm{SITE}\text{:}\\ & \quad \mu_2 \in \mathrm{Reach}(\mu_1)\\ & \quad \Rightarrow \mu_1 \in \mathrm{Reach}(\mu_2)\end{aligned}$$
$$\begin{aligned}(4)\ & \forall \mu_1, \mu_2, \mu_3\text{: } \mu_2 \in \mathrm{Reach}(\mu_1) \wedge \mu_3 \in \mathrm{Reach}(\mu_2)\\ & \quad \Rightarrow \mu_3 \in \mathrm{Reach}(\mu_1)\end{aligned}$$

The first component specifies that any site is reachable to itself. The second rule together with the third rule declares that two adjacent sites are reachable each other if both of them are known by the attacker. The fourth rule says that the relation of Reach is transitive. As the result, given a site named $*$, the function Reach returns all its reachable sites according to the knowledge of the attacker.

In the fourth component the attacker delivers all messages he has in $t_\bullet$ to the reachable sites. Sharing knowledge between attackers is implemented by using the same canonical name $n_\bullet$ and canonical variables $z_\bullet$ and $t_\bullet$ for all of them. This completes the explanation of the formula for the adjusted Dolev-Yao condition.

To establish the correctness of the adjusted Dolev-Yao condition for ABoxed Ambients, we present one assumption and two lemmas.

Assumption 1: Including sites in the well-formed attacker process Q does not increase its computational power.

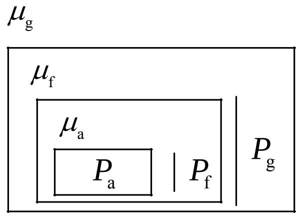
This can be observed by inspecting all primitives that can be used to construct Q. Only input primitives can help it to learn more about environment. Packet ambients together with capabilities and output can be used to actively attack the network. Constructing site ambients in Q, however, just means some packets may be intercepted by the attacker process Q. To learn something about these packets input primitives are the only choices for Q. It means intercepting them can do nothing more. As there is no way for the attacker-constructed sites to control the captured packets, we can not count on the sites attacking the network with the captured packets.

□

Lemma 3: If a packet called μ_a is enclosed by another packet called μ_{f}, and μ_a does not move out of μ_{f}, then any input of ambient μ_{g} that may enclose μ_{f} can not reduce with any output of μ_a, i.e. that

$$\wedge_{k \in A_k} \mu_{\mathrm{g}}[(;x_1,\ldots,x_k)^{\eta}.P \mid \mu_{\mathrm{f}}[\mu_{\mathrm{a}}[<M_1,\ldots,M_k>^{\eta}] \mid Q] \mid P'] \not\mapsto$$

The configuration described above can be presented graphically as:



Proof. The proof is based on the fact that μ_{f} is a packet which has no input and we have no rule for output-to-grandfather, i.e.

$$(;x)^c \mid c[g[<M>^{\uparrow}.P] \mid Q] \not\rightarrow \cdots$$

□

Lemma 4: Including the process that a packet is enclosed by another packet in the well-formed attacker process Q does not increase its computational power; in other words the packets of the attacker process do not have to enclose any other packets and this does not impair the computational power of the attacker process.

Proof. The proof is based on the fact that if the enclosed packet never moves out of its enclosing packet, then by Lemma 1 it can not output any message to any site. Otherwise we can let Q include the two packets directly instead of the case that one encloses another. By this way we are still sure that Q has the same computational power because any packet is "useless" to sites until it is not enclosed by any other packet.

□

Based on Assumption 1 and Lemma 4, we try to set up the relation between adjusted Dolev-Yao attacker and the hardest attacker as below theorem.

Theorem 3: Soundness of adjusted Dolev-Yao attacker

If $(\gamma,\kappa,\rho,\varphi)$ satisfies $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ of type $(\mathcal{N}_{\mathrm{f}},\mathcal{A}_{\mathcal{K}})$ then $(\gamma,\kappa,\rho) \models^* \overline{Q} : \varphi$ for all well-formed processes Q of type $(\mathcal{N}_{\mathrm{f}},\mathcal{A}_{\mathcal{K}})$.

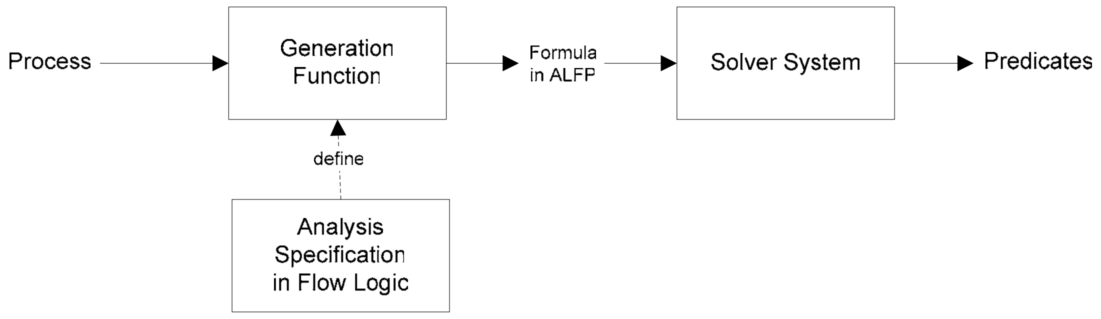
The formal proof is given in Appendix D.2 for your reference.

As one may note, that our attacker has the capability to cache encrypted messages, which are expressed as an ambient named by a private key, and forward them to reachable sites. This capability is necessary for a network attacker but can not be implemented by our calculus. However this limitation should not affect our analysis to get sensible estimate for the purpose of protocol validation since we more concern whether or not the formula $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ captures all necessary capabilities that the attacker are supposed to have.

5 Implementation

In this chapter we aim at developing an automatic tool which can compute our control flow analysis correctly. To achieve that, we need specify the implementation of the analysis specification in order to compute the estimate for a process. For our analysis, specifically, it means that given a process P the implementation should come out the approximation $(\gamma, \kappa, \rho, \varphi)$ such that $(\gamma, \kappa, \rho) \models^{\mu} P : \varphi$.

The overall strategy for implementation is to define a generation function, $\mathcal{G}(P)$ that given an ABoxed Ambients process returns a formula. Some logic solver is then used to compute the estimate predicates for the formula. Here our formula shall be written in Alternation-free Least Fixed Point (ALFP) logic and the design of the tool could be sketched as below



As the figure presents, the initial input of the tool is a process written in ABoxed Ambients which is then translated into ALFP formulae by the generation function. The definition of the function is directed by analysis specifications. Sometimes an analysis specification can not be used directly to define a generation function and several additional steps are required. At last the solver system computes the generated formula and provides the analysis results in the form of predicates.

Many systems can serve for the computation, such as iterative-base worklist algorithm, XSB Prolog v2.6 [21] [20] and Succinct Solver v2.0 [13,12]. A well-designed experiment that compares the Succinct Solver with XSB Protolog can be found at [22]. It shows that the performance of Succinct Solver against XSB Prolog is quite encouraging. Furthermore Succinct Solver may deal with the specification for which XSB Prolog can not produce a solution in a few cases.

We here choose Succinct Solver as our constraint solver to obtain an efficient implementation. Many kinds of applications have been used to validate the robustness of the specification language and to suggest additional features that can be provided by Succinct Solver in order to improve the usability of the tool for even non-expert users. These applications include security properties of Java Card bytecode [23], access control features of Mobile and Discretionary Ambients [3] and so on.

Especially it has been used for the implementation of the control flow analysis of both Boxed Ambients and LySa calculus, which impact the design of our analysis essentially. The strategy they used, therefore, serves as a good reference for implementing our analysis.

Our tool has been implemented in Standard ML of New Jersey and the syntax of ABoxed Ambients is modified a little to deal with some implementation issues, i.e. there is no explicit function defined for canonical operation as its domain is over infinite name space. Moreover we introduce the Indexed ABoxed Ambients for the convenience of declaring multi-sites which run protocols at the same time. All these changes have no semantic consequence and thus do not affect the analysis specification.

5.1 Analysis in ALFP Logic

ALFP logic is a powerful fragment of first-order predicate logic. It extends Horn clauses by allowing both existential and universal quantifications in pre-conditions, negative queries (that is subject to the notion of stratification), disjunctions of pre-conditions and conjunctions of conclusion. In this sub-section we present the necessary steps to transform the analysis into the formula of ALFP logic. That formula serves as the input of Succinct Solver which, in return, outputs the least interpretation of predicates satisfying a given formula.

5.1.1 From Succinct to Verbose

The analysis specified in Table 3-1 and 3-4 is *succinct* as there is a component φ on the right hand side of the judgment, $(\gamma,\kappa,\rho) \models^{\mu} P : \varphi$. As we explained in Chapter 3 the Flow Logic of this style creates a local scope for the analysis component φ. The ALFP logic that is supported by Succinct Solver, however, can not provide scoping mechanisms for predicates. Thus our first step is to ensure that every analysis component has global scope. In other words we need transform the Flow Logic from its *succinct* form into *verbose* form.

The approach for transforming a succinct Flow Logic into its equivalent verbose one was developed in [14]. The essential idea of the technique is to provide a global pointer pointing into the local components by adding labels in the syntax where succinct judgments are used.

But after inspecting the analysis we found that the succinct formulation of the observation predicate φ is actually a global component and can, therefore, be equally rewritten into verbose form. It also means that there is no need for yielding the syntax with labels attached.

The verbose form of the control flow analysis specification is defined in Table 5-1 and the explanation is the same as the one for Table 3-1 and 3-4.

$(\gamma,\kappa,\rho,\varphi)\models^{\mu} (\nu\ n)P$	iff $(\gamma,\kappa,\rho,\varphi)\models^{*} P$
$(\gamma,\kappa,\rho,\varphi)\models^{*} (\nu k\ n)P$	iff $(\gamma,\kappa,\rho,\varphi)\models^{*} P$
$(\gamma,\kappa,\rho,\varphi)\models^{*} 0$	iff true
$(\gamma,\kappa,\rho,\varphi)\models^{*} P_1 \mid P_2$	iff $(\gamma,\kappa,\rho,\varphi)\models^{*} P_1 \wedge (\gamma,\kappa,\rho,\varphi)\models^{*} P_2$
$(\gamma,\kappa,\rho,\varphi)\models^{*} !P$	iff $(\gamma,\kappa,\rho,\varphi)\models^{*} P$
$(\gamma,\kappa,\rho,\varphi)\models^{*} N[P]$	iff $\forall\mu\in\mathcal{N}_\rho(N):\mu\in\gamma(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{\mu} P$
$(\gamma,\kappa,\rho,\varphi)\models^{*}$ in $N.P$	iff $\mathcal{M}_\rho$ (in $N)\subseteq\gamma(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*} P\wedge$
	$\forall$in $\mu\in\mathcal{M}_\rho$ (in $N):\varphi_{\text{in}}(\mu)$
$(\gamma,\kappa,\rho,\varphi)\models^{*}$ out $N.P$	iff $\mathcal{M}_\rho$ (out $N)\subseteq\mathcal{I}(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*} P\wedge$
	$\forall$out $\mu\in\mathcal{M}_\rho$ (out $N):\varphi_{\text{out}}(\mu)$
$(\gamma,\kappa,\rho,\varphi)\models^{*} N.P$	iff $\mathcal{M}_\rho(N)\bigcap M\subseteq\mathcal{I}(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma} P\wedge$
	$\forall$in $\mu\in\mathcal{M}_\rho(N):\varphi_{\text{in}}(\mu)$
	$\forall$out $\mu\in\mathcal{M}_\rho(N):\varphi_{\text{out}}(\mu)$
$(\gamma,\kappa,\rho,\varphi)\models^{*}<M_1,\cdots,M_k>^{\circ}_{\ell}$ [dest $\mathcal{L}$]	iff $\forall v_1,\cdots,v_k:\ \wedge^{k}_{i=1} v_i\in\mathcal{M}_\rho(M_i)$
	$\Rightarrow <v_1,\cdots,v_k>_{\ell}$ [dest $\mathcal{L}]\in\kappa(*)$
$(\gamma,\kappa,\rho,\varphi)\models^{*} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\circ}.P$	iff $<v_1,\cdots,v_k>_{\ell}$ [dest $\mathcal{L}]\in\kappa(*)$:
	$\wedge^{j}_{i=1} v_i\in\mathcal{M}_\rho(M_i)$
	$\Rightarrow \wedge^{k}_{i=j+1} v_i\in\rho(x_i)\wedge$
	$(\neg\text{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L})\Rightarrow(\ell,\varepsilon)\in\varphi)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*} P$
$(\gamma,\kappa,\rho,\varphi)\models^{*}<M_1,\cdots,M_k>^{\text{N}}$	iff $\forall\mu\in\mathcal{N}_\rho(\text{N}):\mu\in\gamma(*)\ \wedge$
	$\forall v_1,\cdots,v_k:\ \wedge^{k}_{i=1} v_i\in\mathcal{M}_\rho(M_i)$
	$\Rightarrow <v_1,\cdots,v_k>_{\varepsilon}$ [dest $\mathcal{D}]\subseteq\kappa(*)$
$(\gamma,\kappa,\rho,\varphi)\models^{*} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{N}_{\ell'}$[orig $\mathcal{L}'].P$	iff $\forall\mu\in\mathcal{N}_\rho(\text{N}):\mu\in\gamma(*)\wedge$
	$\forall <v_1,\cdots,v_k>_{\ell}$ [dest $\mathcal{L}]\in\kappa(\mu)$: $\wedge^{j}_{i=1} v_i\in\mathcal{M}_\rho(M_i)$
	$\Rightarrow \wedge^{k}_{i=j+1} v_i\in\rho(x_i)\wedge$
	$(\neg\text{RM}(\ell,\mathcal{L}',\ell',\mathcal{L})\Rightarrow(\ell,\ell')\in\varphi)\wedge$
	$(\gamma,\kappa,\rho,\varphi)\models^{*} P$
$(\gamma,\kappa,\rho,\varphi)\models^{*}<M_1,\cdots,M_k>^{\uparrow}$	iff $\forall\mu:*\in\gamma(\mu)\ \wedge\forall v_1,\cdots,v_k:\ \wedge^{k}_{i=1} v_i\in\mathcal{M}_\rho(M_i)$
	$\Rightarrow <v_1,\cdots,v_k>_{\varepsilon}$ [dest $\mathcal{D}]\subseteq\gamma(*)$
$(\gamma,\kappa,\rho,\varphi)\models^{*} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\uparrow}.P$	iff $\forall\mu:*\in\gamma(\mu)\wedge$
	$\forall <v_1,\cdots,v_k>_{\ell}$ [dest $\mathcal{L}]\in\kappa(\mu))$: $\wedge^{j}_{i=1} v_i\in\mathcal{M}_\rho(M_i)$
	$\Rightarrow v_{j+1}\in\rho(x_j)\wedge\cdots\wedge v_k\in\rho(x_k)\wedge$
	$(\neg\text{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L})\Rightarrow(\ell,\varepsilon)\in\varphi)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}$ P

Table 5-1: **Verbose formulation of the analysis.**

5.1.2 From Infinite Name Space to Finite One

We defined a canonical function which maps a naming to its canonical version. The domain of this function is an infinite set in principle. But to ensure the analysis specification can be realized we have to define how a canonical naming is calculated given a process. One way is to declare the canonical name for every name or variable explicitly in syntax including free names, restricted names and variables. Then even though two names are different semantically, they may share a same canonical name. The idea is partly realized in the control flow analysis of Boxed Ambients in [3] in which the type of a restricted name is declared in syntax explicitly. Another way to solve the problem is to treat the semantic value as just its canonical one and accordingly any two names different in name are always considered different in analysis. This solution is used by LySa tool [24].

Our solution is somewhat a kind of compromise between them: we choose to declare canonical names for free names and restricted names in syntax as well as treat the variable names as their canonical one in the tool. By this way we keep the flexibility of combining two names (semantically different) into one canonical name. The syntax for the two restriction primitives are extended to be

$$(v\ n, \mu)P \text{ and } (vk\ n, \mu)P$$

where μ is the canonical name of a name *n* is recorded by analysis.

The changes have no semantic consequence: we treat them as original primitives and it simply provides information for the convenience of implementing analysis. We then need another environment function $\Gamma : \mathcal{C} \to \lfloor \mathcal{C} \rfloor$ to store the information. The auxiliary function $\mathcal{N}$ and $\mathcal{M}$ are then refined as

$$\begin{aligned} \mathcal{N}_{\rho,\Gamma}(x) &= \rho(\lfloor x \rfloor) \cap \lfloor \mathcal{C} \rfloor \\ \mathcal{N}_{\rho,\Gamma}(n) &= \{\lfloor n \rfloor\} \text{ where } \lfloor n \rfloor = \Gamma(n) \end{aligned}$$

$$\begin{aligned} \mathcal{M}_{\rho,\Gamma}(\text{in } N) &= \{\text{in } \mu \mid \mu \in \mathcal{N}_{\rho}(N)\} \\ \mathcal{M}_{\rho,\Gamma}(\text{out } N) &= \{\text{in } \mu \mid \mu \in \mathcal{N}_{\rho}(N)\} \\ \mathcal{M}_{\rho,\Gamma}(x) &= \rho(\lfloor x \rfloor) \\ \mathcal{M}_{\rho,\Gamma}(n) &= \{\lfloor n \rfloor\} \text{ where } \lfloor n \rfloor = \Gamma(n) \end{aligned}$$

As the information contained by Γ is acquired by the syntax we would rather consider it as an environment than an analysis component and, therefore the judgment of analysis is defined as:

$$(\gamma, \kappa, \rho, \varphi) \models_{\Gamma}^{\mu} P$$

The canonical name for the free names of a process can be declared by restriction i.e.

$$(\nu\ \mathrm{fn}_1, \mu_{\mathrm{f}_1})...(\nu\ \mathrm{fn}_k, \mu_{\mathrm{f}_k})P$$

Surely the rewritten process as a whole has no free variable but we can always pick up the interested part, i.e. *P*, and calculate its free name separately. The added restriction serves for providing environment information for the analysis.

The process is now a little cumbersome. But soon we found that this style is worthy when attackers are considered. For example, for the process $P \mid \bullet$ we shall rewrite it to be

$$(\nu\ \mathrm{fn}_1, \mu_{\mathrm{f}_1})...(\nu\ \mathrm{fn}_k, \mu_{\mathrm{f}_k})(P \mid \bullet)$$

By this way, all names available to attackers become clear. As the complexity introduced by the hierarchical structure of ambients, it becomes important that we are able to clearly present the initial knowledge of the attacker in order to check if he acquires more knowledge than expected.

We have given a specific definition for the canonical operation. There is, however, another issue about infinite set: The sets of site names and packet names are infinite and are impossible to be listed in a concrete implementation. We thus turn to finite sets. There are at least two ways for achieving that goal. The first way is to pre-define a finite set of site names and then others simply belong to packet names. The limitation is that we can not name a site any name as we like. The second way, similar to the solution for canonical names, lets syntax declare that which name is for site. This is feasible since we now restrict all names and thus we can declare a name is for site when we introduce it. We choose the second way as it only causes little extension to the syntax and in return we can name sites in a more friendly and understandable style. We add a new restriction primitive

$$(\nu s\ n, \mu)$$

which claims that *n* is a site name. Again from semantic view, this primitive is just $(\nu\ n, \mu)$. But it benefits our implementation in defining the set of site names used in a process. Now the names restricted by $(\nu\ n, \mu)$ and $(\nu k\ n, \mu)$ are considered as packet names. Here we do not extend the secret restriction $(\nu k\ n, \mu)$ to be $(\nu ks\ n, \mu)$ since a secret name usually is used as packet name. Two environment components $\mathcal{S} : \lfloor \mathcal{C} \rfloor \rightarrow \{\text{true,false}\}$ and $\mathcal{P} : \lfloor \mathcal{C} \rfloor \rightarrow \{\text{true,false}\}$ is then needed and the judgment is thus modified as

$$(\gamma, \kappa, \rho, \varphi) \models^{\mu}_{\Gamma,\mathcal{S},\mathcal{P}} P$$

The resulting analysis specification is listed in Table 5-2. Compared with those of Table 5-1, the predicates of the three kinds of restrictions are modified or added for defining the canonical operation and the set of site or packet names.

$$(\gamma,\kappa,\rho,\varphi)\models^{\mu}_{\Gamma,\mathcal{S},\mathcal{P}} (v\ n,\mu)P \quad \text{iff}\ (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma[n\to\mu],\mathcal{S},\mathcal{P}[\mu\to\text{true}]} P$$

$$(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (vk\ n,\mu)P \quad \text{iff}\ (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma[n\to\mu],\mathcal{S},\mathcal{P}[\mu\to\text{true}]} P$$

$$(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (vs\ n,\mu)P \quad \text{iff}\ (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma[n\to\mu],\mathcal{S}[\mu\to\text{true}],\mathcal{P}} P$$

$$(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} 0 \quad \text{iff}\ \text{true}$$

$$(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P_1 | P_2 \quad \text{iff}\ (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P_1 \wedge (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P_2$$

$$(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} !P \quad \text{iff}\ (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P$$

$$(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} N[P] \quad \text{iff}\ \forall\mu\in\mathcal{N}_{\Gamma,\rho}(N):\mu\in\gamma(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{\mu}_{\Gamma,\mathcal{S},\mathcal{P}} P$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} \text{in}\ N.P \quad & \text{iff}\ \mathcal{M}_{\Gamma,\rho}(\text{in}\ N)\subseteq\gamma(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P\wedge\\ & \forall\text{in}\ \mu\in\mathcal{M}_{\Gamma,\rho}(\text{in}\ N):\varphi_{\text{in}}(\mu)\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} \text{out}\ N.P \quad & \text{iff}\ \mathcal{M}_{\Gamma,\rho}(\text{out}\ N)\subseteq\mathcal{I}(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P\wedge\\ & \forall\text{out}\ \mu\in\mathcal{M}_{\Gamma,\rho}(\text{out}\ N):\varphi_{\text{out}}(\mu)\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} N.P \quad & \text{iff}\ \mathcal{M}_{\Gamma,\rho}(N)\bigcap M\subseteq\mathcal{I}(*)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P\wedge\\ & \forall\text{in}\ \mu\in\mathcal{M}_{\Gamma,\rho}(N):\varphi_{\text{in}}(\mu)\\ & \forall\text{out}\ \mu\in\mathcal{M}_{\Gamma,\rho}(N):\varphi_{\text{out}}(\mu)\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} < M_1,\cdots,M_k >^{\circ}_{\ell} [\text{dest}\ \mathcal{L}]\ & \text{iff}\ \forall v_1,\cdots,v_k:\ \wedge^{k}_{i=1} v_i\in\mathcal{M}_{\Gamma,\rho}(M_i)\\ & \Rightarrow <v_1,\cdots,v_k>_{\ell} [\text{dest}\ \mathcal{L}]\in\kappa(*)\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\circ}.P\ & \text{iff} <v_1,\cdots,v_k>_{\ell} [\text{dest}\ \mathcal{L}]\in\kappa(*):\ \wedge^{j}_{i=1} v_i\in\mathcal{M}_{\Gamma,\rho}(M_i)\\ & \Rightarrow \wedge^{k}_{i=j+1} v_i\in\rho(x_i)\wedge\\ & (\neg\text{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L})\Rightarrow(\ell,\varepsilon)\in\varphi)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} < M_1,\cdots,M_k >^{\text{N}}\ & \text{iff}\ \forall\mu\in\mathcal{N}_{\Gamma,\rho}(\text{N}):\mu\in\gamma(*)\ \wedge\\ & \forall v_1,\cdots,v_k:\ \wedge^{k}_{i=1} v_i\in\mathcal{M}_{\Gamma,\rho}(M_i)\\ & \Rightarrow <v_1,\cdots,v_k>_{\varepsilon} [\text{dest}\ \mathcal{D}]\subseteq\kappa(*)\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{N}_{\ell'}[\text{orig}\ \mathcal{L}'].P\ & \text{iff}\ \forall\mu\in\mathcal{N}_{\Gamma,\rho}(\text{N}):\mu\in\gamma(*)\wedge\\ & \forall < v_1,\cdots,v_k >_{\ell} [\text{dest}\ \mathcal{L}]\in\kappa(\mu):\ \wedge^{j}_{i=1} v_i\in\mathcal{M}_{\Gamma,\rho}(M_i)\\ & \Rightarrow \wedge^{k}_{i=j+1} v_i\in\rho(x_i)\wedge\\ & (\neg\text{RM}(\ell,\mathcal{L}',\ell',\mathcal{L})\Rightarrow(\ell,\ell')\in\varphi)\wedge\\ & (\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} < M_1,\cdots,M_k >^{\uparrow}\ & \text{iff}\ \forall\mu:*\in\gamma(\mu)\ \wedge\forall v_1,\cdots,v_k:\ \wedge^{k}_{i=1} v_i\in\mathcal{M}_{\Gamma,\rho}(M_i)\\ & \Rightarrow <v_1,\cdots,v_k>_{\varepsilon} [\text{dest}\ \mathcal{D}]\subseteq\gamma(*)\end{aligned}$$

$$\begin{aligned}(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\uparrow}.P\ & \text{iff}\ \forall\mu:*\in\gamma(\mu)\wedge\\ & \forall < v_1,\cdots,v_k >_{\ell} [\text{dest}\ \mathcal{L}]\in\kappa(\mu)):\ \wedge^{j}_{i=1} v_i\in\mathcal{M}_{\Gamma,\rho}(M_i)\\ & \Rightarrow v_{j+1}\in\rho(x_j)\wedge\cdots\wedge v_k\in\rho(x_k)\wedge\\ & (\neg\text{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L})\Rightarrow(\ell,\varepsilon)\in\varphi)\wedge(\gamma,\kappa,\rho,\varphi)\models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} \text{P}\end{aligned}$$

Table 5-2: Analysis with new environment functions Γ, $\mathcal{S}$ and $\mathcal{P}$.

5.1.3 Encoding Annotations

The Communication component need record a set of crypto-points which are used to declare intended encryption or decryption points. Because Succinct Solver works on unstructured universes, we can not use it to represent these sets directly. Therefore we need a further transformation for the analysis in Table 5-2. The transformation is standard [14, 24] and is conducted by introducing a unique label into the syntax at every set of crypto-points and add a global analysis component $\delta: Lab \rightarrow \mathcal{P}(\mathcal{D})$. The function of the component is to store the sets of crypto-points for each label. The communication analysis component $\kappa: \lfloor \mathcal{C} \rfloor \rightarrow \mathcal{P}((\lfloor \mathcal{C} \rfloor \cup \lfloor \mathcal{M} \rfloor)^*)$ is accordingly modified to be

$$\kappa: \lfloor \mathcal{C} \rfloor \rightarrow \mathcal{P}((\lfloor \mathcal{C} \rfloor \cup \lfloor \mathcal{M} \rfloor)^* \times \mathrm{Lab}^2)$$

In Table 5-3 (presented on next page) we list the modified analysis for all communication primitives. For other rules in Table 5-2 we simply add a new analysis component δ on the left right hand side of the judgment. The explanation to the rules in Table 5-3 is given below.

For the primitive of local output which declares crypto-points explicitly, we first introduce a unique label called l_c into the syntax of local output, i.e.

$$< M_1, \cdots, M_k >^{\circ}_{\ell} [\text{dest } \mathcal{L}^{l_c}]$$

The analysis then generate a set of predicates together with the estimate of communication

$$\wedge_{c \in \mathcal{L}} c \in \delta(l_c) \wedge (\forall v_1, \cdots, v_k : \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\mathrm{T},\rho}(M_i)) \Rightarrow < v_1, \cdots, v_k, \ell, l_c > \in \gamma(*)$$

For the local input we check if the available message is a plain text. Remember that we use trivial crypto-point ε and the whole set $\mathcal{D}$ to express plain message. As $\mathcal{D}$ is theoretically infinite, however, we can not list all its members. This can be solved by introducing a special label, AW (anywhere): If the label pointing into destinations of an output is declared as AW, then the check of decryption point in the input side always succeeds. Similarly if the label for the origins of an input is AW then we omit checking the encryption point.

The parent-to-child output predicate adds the special label AW into tuples of communication estimate silently in analysis. The predicate for the parent-child input extends the syntax with the label l_c that is adopted to generate a set of predicates like those created in local output. To handle the case of plain message, it uses three clauses

$$(l_c \neq \mathrm{AW} \wedge l'_c \neq \mathrm{AW}) \Rightarrow (\ell' \in \delta(l_c) \vee \ell \in \delta(l'_c)) \Rightarrow (\ell', \ell) \in \varphi \wedge$$
$$(l_c \neq \mathrm{AW} \wedge l'_c = \mathrm{AW}) \Rightarrow (\ell' \notin \delta(l_c) \Rightarrow (\ell', \varepsilon) \in \varphi) \wedge$$
$$(l_c = \mathrm{AW} \wedge l'_c \neq \mathrm{AW}) \Rightarrow (\ell \notin \delta(l'_c) \Rightarrow (\varepsilon, \ell) \in \varphi)$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} < M_1,\cdots,M_k >^{\circ}_{\ell} [\text{dest } \mathcal{L}^{l_c}]$$

$$\text{iff } \wedge_{c\in\mathcal{L}}\, c \in \delta(l_c) \wedge (\forall v_1,\cdots,v_k\!:\ \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\Gamma,\rho}(M_i))$$

$$\Rightarrow <v_1,\cdots,v_k,\ell,l_c> \in \gamma(*)$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\circ}.P$$

$$\text{iff } <v_1,\cdots,v_k,\ell,l_c> \in \gamma(*)\text{:}$$

$$\wedge_{i=1}^{j} v_i \in \mathcal{M}_{\Gamma,\rho}(M_i)$$

$$\Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(x_i) \wedge$$

$$(l_c \neq \text{AW} \Rightarrow \varepsilon \notin \delta(l_c) \Rightarrow (\ell,\varepsilon) \in \varphi) \wedge$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} < M_1,\cdots,M_k >^{N} \text{ iff } \forall \mu \in \mathcal{N}_{\Gamma,\rho}(\text{N}) : \mu \in \gamma(*) \ \wedge$$

$$\forall v_1,\cdots,v_k\!:\ \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\Gamma,\rho}(M_i)$$

$$\Rightarrow\ <v_1,\cdots,v_k,\varepsilon,\text{AW}> \subseteq \gamma(*)$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{N}_{\ell'}[\text{orig } \mathcal{L}'^{l'_c}].P$$

$$\text{iff } \wedge_{c'\in\mathcal{L}'}\, c' \in \delta(l'_c) \wedge (\forall \mu \in \mathcal{N}_{\Gamma,\rho}(\text{N}) : \mu \in \gamma(*) \wedge$$

$$\forall < v_1,\cdots,v_k,\ell,l_c > \in \kappa(\mu)\text{:}\ \wedge_{i=1}^{j} v_i \in \mathcal{M}_{\Gamma,\rho}(M_i))$$

$$\Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(x_i) \wedge$$

$$(l_c \neq \text{AW} \wedge l'_c \neq \text{AW}) \Rightarrow (\ell' \in \delta(l_c) \vee \ell \in \delta(l'_c)) \Rightarrow (\ell',\ell) \in \varphi \wedge$$

$$(l_c \neq \text{AW} \wedge l'_c\text{=AW}) \Rightarrow (\ell' \notin \delta(l_c) \Rightarrow (\ell',\ell) \in \varphi) \wedge$$

$$(l_c\text{=AW} \wedge l'_c \neq \text{AW}) \Rightarrow (\ell \notin \delta(l'_c) \Rightarrow (\ell',\ell) \in \varphi) \wedge$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} < M_1,\cdots,M_k >^{\uparrow} \text{ iff } \forall \mu : * \in \gamma(\mu) \ \wedge \forall v_1,\cdots,v_k\!:\ \wedge_{i=1}^{k} v_i \in \mathcal{M}_{\Gamma,\rho}(M_i)$$

$$\Rightarrow <v_1,\cdots,v_k,\varepsilon,\text{AW}> \subseteq \kappa(*)$$

$$(\gamma,\kappa,\rho,\varphi) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} (M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\uparrow}.P$$

$$\text{iff } \forall \mu : * \in \gamma(\mu) \wedge$$

$$\forall < v_1,\cdots,v_k,\ell,l_c > \in \mathcal{C}(\mu))\text{:}\ \wedge_{i=1}^{j} v_i \in \mathcal{M}_{\Gamma,\rho}(M_i)$$

$$\Rightarrow v_{j+1} \in \rho(x_j) \wedge \cdots \wedge v_k \in \rho(x_k) \wedge$$

$$(l_c \neq \text{AW}) \Rightarrow (\varepsilon \notin \delta(l_c) \Rightarrow (\varepsilon,\ell) \in \varphi) \wedge$$

$$(\gamma,\kappa,\rho,\varphi,\delta) \models^{*}_{\Gamma,\mathcal{S},\mathcal{P}} P$$

Table 5-3: **Encoding annotations in the analysis**

to determine which crypto-points should be checked. If both two labels, l_c and l'_c, are AW, no label pair can be added into φ.

The predicate for child-to-parent input and output are modified in the similar way presented above. This completes the whole transformation.

5.1.4 Generating ALFP Logic Formulae

We are now ready to define a generation function $\mathcal{G}$, which takes a process as input and returns an ALFP formula. In this sub-section, we first give a brief introduction to ALFP logic and the Succinct Solver. Then the generation function for the analysis is declared. Last we present the ALFP formula for the attacker process.

5.1.4.1 ALFP Logic and Succinct Solver

Alternation-free Least Fixed Point logic (ALFP) is a fragment of first order predicate logic. An ALFP formula *cl* is consists of a fixed countable set $\mathcal{X}$ of variables *x*, a finite set $\mathcal{C}$ of constant symbols *c*, a finite ranked alphabet $\mathcal{R}$ of predicate symbol *R*, a finite set $\mathcal{F}$ of function symbols *f* and pre-conditions *pre* , following the grammar in Table 5-4.

t	$::=$	$c \mid x \mid f(t_1,...,t_k)$
pre	$::=$	$R(t_1,...,t_k) \mid \neg R(t_1,...,t_k) \mid pre_1 \wedge pre_2$
	$\mid$	$pre_1 \vee pre_2 \mid \exists x : pre \mid t_1 = t_2 \mid t_1 \neq t_2$
	$\mid$	$1 \mid 0$
cl	$::=$	$R(t_1,...,t_k) \mid 1 \mid cl_1 \wedge cl_2$
	$\mid$	$pre \Rightarrow cl \mid \forall x : cl$

Table 5-4: **Abstract syntax of ALFP clauses**

As presented in Table 5-4, *R* is a *k*-ary predicate symbol for $k \geq 0$, t_1, …, t_k denote arbitrary variables and 1 is always true clause. We call R(…) and ¬R(…) occurring in pre-conditions *queries* and *negative queries* respectively whereas the other occurrences are called *assertions* of the predicate *R*.

The notion of stratification is introduced by the Succinct Solver to ensure the negation can be handled conveniently. A clause *cl* is an alternation-free Least Fixpoint formula if it has the form $cl = cl_1 \wedge cl_2 \wedge ... \wedge cl_n$ and there is a function *rank*: $\mathcal{R} \rightarrow \mathbb{N}$ such that for all *j*=1, …, *n*, the following properties hold:

(1) all predicates of assertions in cl_j have rank *j*;

(2) all predicates of queries in cl_j have ranks at most *j*;

(3) all predicates of negated queries in cl_j have ranks strictly less than *j*.

We illustrate the concept of stratification by the following example.

Example 5.1 Given two 1-ary predicates *P* and Q, then:

1. The clause $Q(a) \wedge (\forall x : \neg Q(x) \rightarrow P(x))$ is stratified, where *rank*(Q) = 1 and *rank*(*P*) = 2.

2. The clause $(\forall x : P(x) \rightarrow Q(x)) \wedge (\forall x : \neg Q(x) \rightarrow P(x))$ is not stratified since it is impossible to have *rank*(*P*) ≤ *rank*(Q) as well as *rank*(Q) < *rank*(*P*).

In the current version of the solver, the stratification is implicitly given by the user. That means users are responsible for ensuring that the predicate occurring in the negative query has been defined in the lower strata, and will not appear anymore thereafter.

Given a non-empty and countable universe $\mathcal{U}$, and interpretations ρ and σ for predicate symbols and free names respectively, we define the satisfaction relations

$$(\rho,\sigma) \models pre \text{ and } (\rho,\sigma) \models cl$$

for pre-conditions and clauses in a straightforward manner as shown in Table 5-5.

$(\rho,\sigma) \models R(x_1,...,x_k)$	iff	$(\sigma(x_1),...,\sigma(x_k)) \in \rho(R)$
$(\rho,\sigma) \models \neg R(x_1,...,x_k)$	iff	$(\sigma(x_1),...,\sigma(x_k)) \notin \rho(R)$
$(\rho,\sigma) \models pre_1 \wedge pre_2$	iff	$(\rho,\sigma) \models pre_1$ and $(\rho,\sigma) \models pre_2$
$(\rho,\sigma) \models pre_1 \vee pre_2$	iff	$(\rho,\sigma) \models pre_1$ or $(\rho,\sigma) \models pre_2$
$(\rho,\sigma) \models \exists x : pre$	iff	$(\rho,\sigma[x \mapsto a]) \models pre$ for some $a \in \mathcal{U}$
$(\rho,\sigma) \models \forall x : pre$	iff	$(\rho,\sigma[x \mapsto a]) \models pre$ for all $a \in \mathcal{U}$

$(\rho,\sigma) \models R(x_1,...,x_k)$	iff	$(\sigma(x_1),...,\sigma(x_k)) \in \rho(R)$
$(\rho,\sigma) \models 1$	iff	always
$(\rho,\sigma) \models cl_1 \wedge cl_2$	iff	$(\rho,\sigma) \models cl_1$ and $(\rho,\sigma) \models cl_2$
$(\rho,\sigma) \models pre \Rightarrow cl$	iff	$(\rho,\sigma) \models cl$ whenever $(\rho,\sigma) \models pre$
$(\rho,\sigma) \models \forall x : cl$	iff	$(\rho,\sigma[x \mapsto a]) \models cl$ for all $a \in \mathcal{U}$

Table 5-5: Semantics of ALFP clauses

We shall consider the free variables occurring in a formula as constant symbols or atoms from the finite universe $\mathcal{U}$. Here we write $\rho(R)$ for the set of *k*-tuples $(a_1,...,a_k)$ from $\mathcal{U}$ associated with the *k*-ary predicate *R*. We also write $\sigma(x)$ for the atom of $\mathcal{U}$ bound to *x* and finally $\sigma[x \mapsto a]$ stands for the mapping that is as σ except that *x* is mapped to a.

As we mentioned at the beginning of this chapter, the Succinct Solver takes the ALFP formula as input and compute the least solution for the analysis which enjoys *Moore family* property. In the previous sections, we have transformed our analysis to work on global analysis components over finite domains. We are now ready to express the analysis in ALFP formulae.

5.1.4.2 The Generation Function for the Analysis

We first consider the translation of the analysis components. The component $\gamma{:}\lfloor \mathcal{C} \rfloor \to \mathcal{P}(\lfloor \mathcal{C} \rfloor \cup \lfloor \mathrm{Cap} \rfloor)$ can be encoded isomorphically as a binary predicate $\gamma \in \mathcal{P}$ $(\lfloor \mathcal{C} \rfloor \times (\lfloor \mathcal{C} \rfloor \cup \lfloor \mathrm{Cap} \rfloor))$. Following the similar idea we represent the components $\rho{:}\lfloor \mathcal{X} \rfloor \to \mathcal{P}(\lfloor \mathcal{C} \rfloor \cup \lfloor \mathrm{Cap} \rfloor)$ and $\kappa{:}\lfloor \mathcal{C} \rfloor \to \mathcal{P}((\lfloor \mathcal{C} \rfloor \cup \lfloor \mathrm{Cap} \rfloor)^* \times \mathrm{Lab}^2)$ to be $\rho \in \mathcal{P}(\lfloor \mathcal{X} \rfloor \times$ $(\lfloor \mathcal{C} \rfloor \cup \lfloor \mathrm{Cap} \rfloor))$ and $\kappa \in \mathcal{P}(\lfloor \mathcal{C} \rfloor \times ((\lfloor \mathcal{C} \rfloor \cup \lfloor \mathrm{Cap} \rfloor)^* \times \mathrm{Lab}^2))$ respectively.

For the analysis considered in Table 5-3 and the non-communication related rules in Table 5-2, the following transformations are suggested at the same time:

- Set membership such as $\mu' \in \gamma(\mu)$ is written as $\gamma(\mu, \mu')$
- Subset relations such as $\gamma(\mu) \subseteq \gamma(\mu')$ are coded by explicitly quantifying the elements in the first set: $\forall e : \gamma(\mu, e) \Rightarrow \gamma(\mu', e)$.
- Quantifications in the form of $\forall e \in \gamma(\mu)$: ... are expanded as $\forall e, \mu : \gamma(\mu, e) \Rightarrow \ldots$

We define a generation function $\mathcal{G}$ that takes a process as input and returns an ALFP formula. The strategy is to take each rule from the analysis and translate them into an ALFP formula equivalent to the hypothesis of the rules. The translation is relevant easy since the analysis has been refined over a finite and unstructured universe.

The generation function takes the form $\mathcal{G}(P, *)$ where $*$ is the enclosing ambient name of the process P. We specify the generation function in Table 5-6 and Table 5-7. The auxiliary functions $\mathcal{N}_{\Gamma,\rho}$ and $\mathcal{M}_{\Gamma,\rho}$ are written as auxiliary generation function $\mathcal{G}_{\mathcal{N}}$ and $\mathcal{G}_{\mathcal{M}}$ listed below.

$$\begin{array}{rcl}
\mathcal{G}_{\mathcal{N}}(n) & = & \forall \mu : \Gamma(n, \mu) \Rightarrow \mathcal{N}(n, \mu) \\
\mathcal{G}_{\mathcal{N}}(x) & = & \forall \mu : \rho(x, \mu) \wedge \\
 & & \forall t : \mu \neq \mathrm{in}(t) \wedge \mu \neq \mathrm{out}(t) \\
 & & \Rightarrow \mathcal{N}(x, \mu) \\
\hline
\mathcal{G}_{\mathcal{M}}(\mathrm{in}\, N) & = & \mathcal{G}_{\mathcal{N}}(N) \wedge \\
 & & \forall \mu : \mathcal{N}(N, \mu) \Rightarrow \mathcal{M}(\mathrm{in}(N), \mathrm{in}(\mu)) \\
\mathcal{G}_{\mathcal{M}}(\mathrm{out}\, N) & = & \mathcal{G}_{\mathcal{N}}(N) \wedge \\
 & & \forall \mu : \mathcal{N}(N, \mu) \Rightarrow \mathcal{M}(\mathrm{out}(N), \mathrm{out}(\mu)) \\
\mathcal{G}_{\mathcal{M}}(N) & = & \mathcal{G}_{\mathcal{N}}(N)
\end{array}$$

As one can see, the transformation is quite straightforward with respect to the original function definition.

$\mathcal{G}(0,*)$	$=$	true
$\mathcal{G}((\nu\ n,\mu)P,*)$	$=$	$\Gamma(n,\mu) \wedge \mathcal{G}(P,*)$
$\mathcal{G}((\nu k\ n,\mu)P,*)$	$=$	$\Gamma(n,\mu) \wedge \mathcal{G}(P,*)$
$\mathcal{G}((\nu s\ n,\mu)P,*)$	$=$	$\Gamma(n,\mu) \wedge \mathcal{S}(n) \wedge \mathcal{G}(P,*)$
$\mathcal{G}(P_1 \mid P_2,*)$	$=$	$\mathcal{G}(P_1,*) \wedge \mathcal{G}(P_2,*)$
$\mathcal{G}(!P,*)$	$=$	$\mathcal{G}(P,*)$
$\mathcal{G}(!P,*)$	$=$	$\mathcal{G}(P,*)$
$\mathcal{G}(N[P],*)$	$=$	$\mathcal{G}_{\mathcal{N}}(N) \wedge$
		$\forall \mu\colon \mathcal{N}(N,\mu) \Rightarrow \gamma(*,\mu) \wedge \mathcal{G}(P,*)$
$\mathcal{G}(\text{in } N.P,*)$	$=$	$\mathcal{G}_{\mathcal{M}}(\text{in } N) \wedge$
		$\forall t_1\colon \mathcal{M}(\text{in}(N),t_1) \Rightarrow \gamma(*,t_1) \wedge$
		$\forall \mu,t_2\colon \mathcal{M}(\text{in}(N),t_2) \wedge (t_2 = \text{in } \mu)$
		$\Rightarrow \varphi_{\text{in}}(\mu,t_2) \wedge \mathcal{G}(P,*)$
$\mathcal{G}(\text{out } N.P,*)$	$=$	$\mathcal{G}_{\mathcal{M}}(\text{out}(N)) \wedge$
		$\forall t_1\colon \mathcal{M}(\text{out}(N),t_1) \Rightarrow \gamma(*,t_1) \wedge$
		$\forall \mu,t_2\colon \mathcal{M}(\text{out}(N),t_2) \wedge (t_2 = \text{out}(\mu))$
		$\Rightarrow \varphi_{\text{out}}(\mu,t_2) \wedge \mathcal{G}(P,*)$
$\mathcal{G}(N.P,*)$	$=$	$\mathcal{G}_{\mathcal{M}}(N) \wedge$
		$\forall \mu_1,t_1\colon \mathcal{M}(N,t_1) \wedge (t_1 = \text{in}(\mu_1) \vee t_1 = \text{out}(\mu_1)) \Rightarrow \gamma(*,t_1) \wedge$
		$\forall \mu_2,t_2\colon \mathcal{M}(N,t_2) \wedge (t_2 = \text{in}(\mu_2)) \Rightarrow \varphi_{\text{in}}(\mu_2,t_2) \wedge$
		$\forall \mu_3,t_3\colon \mathcal{M}(N,t_3) \wedge (t_3 = \text{in}(\mu_3)) \Rightarrow \varphi_{\text{out}}(\mu_3,t_3) \wedge \mathcal{G}(P,*)$

Table 5-6: The generation function (1).

We still use closure condition φ_{in} and φ_{out} to keep our presentation compact. Their predicates are then transformed as

$$
\begin{aligned}
\varphi_{\text{in}}(\mu,t) \;=\; & \forall \mu_{\text{a}},\mu_{\text{p}} : \gamma(\mu_{\text{a}},t) \wedge \\
& \text{Pack}(\mu_{\text{a}}) \wedge \\
& \gamma(\mu_{\text{p}},\mu_{\text{a}}) \wedge \\
& \gamma(\mu_{\text{p}},\mu) \Rightarrow \gamma(\mu,\mu_a) \\
\varphi_{\text{out}}(\mu,t) \;=\; & \forall \mu_{\text{a}},\mu_{\text{g}} : \gamma(\mu_{\text{a}},t) \wedge \\
& \text{Pack}(\mu_{\text{a}}) \wedge \\
& \gamma(\mu,\mu_{\text{a}}) \wedge \\
& \gamma(\mu_{\text{g}},\mu) \Rightarrow \gamma(\mu_g,\mu_a)
\end{aligned}
$$

The generation function declared in Table 5-6 and 5-7 is isomorphically transformed from analysis rules. In most cases, such transformation is quite straightforward. But still we need be very careful of the negation used in the rules: To keep the clarity of the presentation, the definition of the generation function is not strictly compliant with

$$
\begin{array}{lcl}
\mathcal{G}(< M_1,\cdots,M_k >_{\ell}^{\circ} [\text{dest } \mathcal{L}^{l_c}],*) & = & \wedge_{c\in\mathcal{L}}\, \delta(l_c,c) \wedge \wedge_{i=1}^{k} \mathcal{G}_{\mathcal{M}}(M_i) \wedge \\
 & & \forall v_1,\cdots,v_k:\ \wedge_{i=1}^{k} \mathcal{M}(M_i,v_i) \Rightarrow \gamma(*,v_1,\cdots,v_k,\ell,l_c) \\
\mathcal{G}((M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\circ}.P,*) & = & \wedge_{i=1}^{j} \mathcal{G}_{\mathcal{M}}(M_i) \wedge \\
 & & \forall v_1,\cdots,v_k,\ell,l_c{:}\gamma(*,v_1,\cdots,v_k,\ell,l_c) \\
 & & \Rightarrow \wedge_{i=1}^{j}\ \mathcal{M}(M_i,v_i\ ) \\
 & & \Rightarrow \wedge_{i=j+1}^{k} \rho(x_i,v_i) \wedge \\
 & & \quad (l_c \neq \text{AW} \Rightarrow !\delta(l_c,\varepsilon) \Rightarrow \varphi(\ell,\varepsilon) \wedge \mathcal{G}(P,*)) \\
\mathcal{G}(< M_1,\cdots,M_k >^{N},*) & = & \mathcal{G}_{\mathcal{N}}(N) \wedge \wedge_{i=1}^{k} \mathcal{G}_{\mathcal{M}}(M_i) \wedge \\
 & & (\forall \mu : \mathcal{N}(\text{N},\mu) \wedge \gamma(*,\mu)\ \wedge \\
 & & \forall v_1,\cdots,v_k:\ \wedge_{i=1}^{k} \mathcal{M}(M_i,v_i) \\
 & & \Rightarrow\ \gamma(*,v_1,\cdots,v_k,\varepsilon,\text{AW})) \\
\mathcal{G}((M_1,\cdots,M_j;x_{j+1},\cdots,x_k)_{\ell'}^{N}[\text{orig } \mathcal{L}'^{l_c'}].P,*) & = & \\
 & & \mathcal{G}_{\mathcal{N}}(N) \wedge \wedge_{i=1}^{j} \mathcal{G}_{\mathcal{M}}(M_i) \wedge \\
 & & \wedge_{c'\in\mathcal{L}'}\, c' \in \delta(l_c') \wedge (\forall \mu : \mathcal{N}(\text{N},\mu) \wedge \gamma(*,\mu) \\
 & & \Rightarrow \forall v_1,\cdots,v_k,\ell,l_c : \kappa(\mu,v_1,\cdots,v_k,\ell,l_c)\ \wedge \wedge_{i=1}^{j} \mathcal{M}(M_i,v_i) \\
 & & \Rightarrow \wedge_{i=j+1}^{k} \rho(x_i,v_i) \wedge \\
 & & \quad (l_c \neq \text{AW} \wedge l_c' \neq \text{AW}) \Rightarrow (!\delta(l_c,\ell') \vee !\delta(l_c',\ell)) \Rightarrow \varphi(\ell',\ell) \wedge \\
 & & \quad (l_c \neq \text{AW} \wedge l_c'{=}\text{AW}) \Rightarrow (!\delta(l_c,\ell') \Rightarrow (\ell',\varepsilon) \in \varphi) \wedge \\
 & & \quad (l_c{=}\text{AW} \wedge l_c' \neq \text{AW}) \Rightarrow (\delta(l_c',\ell) \Rightarrow (\varepsilon,\ell) \in \varphi) \wedge \\
 & & \wedge\, \mathcal{G}(P,*)) \\
\mathcal{G}(< M_1,\cdots,M_k >^{\uparrow},*) & = & \wedge_{i=1}^{k} \mathcal{G}_{\mathcal{M}}(M_i) \wedge (\forall \mu : \gamma(\mu,*) \wedge \\
 & & \forall v_1,\cdots,v_k:\ \wedge_{i=1}^{k} \mathcal{M}(M_i,v_i) \\
 & & \Rightarrow \kappa(*,v_1,\cdots,v_k,\varepsilon,\text{AW})) \\
\mathcal{G}((M_1,\cdots,M_j;x_{j+1},\cdots,x_k)^{\uparrow}.P,*) & = & \wedge_{i=1}^{j} \mathcal{G}_{\mathcal{M}}(M_i) \wedge \\
 & & (\forall \mu : \gamma(\mu,*) \wedge \\
 & & \quad \forall\forall v_1,\cdots,v_k,\ell,l_c : \kappa(\mu,v_1,\cdots,v_k,\ell,l_c)\ \wedge \wedge_{i=1}^{j} \mathcal{M}(M_i,v_i) \\
 & & \Rightarrow \rho(x_j,v_{j+1}) \wedge \cdots \wedge \rho(x_k,v_k) \wedge \\
 & & (l_c \neq \text{AW}) \Rightarrow (!\delta(l_c,\varepsilon) \Rightarrow \varphi(\varepsilon,\ell) \wedge \mathcal{G}(P,*))
\end{array}
$$

Table 5-7: **The generation function (2).**

the stratified negation required by the Succinct Solver. These negation operations can be found in Table 5-7. To ensure our implementation is compliant with the requirement of the stratification, we use a technique called pending query introduced in [24] to solve the stratification issue. We replace every clause such as

$$(!\delta(l_c,\ell') \vee !\delta(l_c',\ell)) \Rightarrow \varphi(\ell',\ell)$$

with the clause

$$PQF(l_c, \ell', l_c', \ell)$$

At the end of all clauses, the below clause is inserted to complete the query

$$\forall l_c, \ell', l_c', \ell : PQF(l_c, \ell', l_c', \ell) \Rightarrow$$
$$((!\delta(l_c, \ell') \vee !\delta(l_c', \ell)) \Rightarrow \varphi(\ell', \ell))$$

With this further refinement, our implementation are expected to strictly meet the stratification requirement of the Succinct Solver.

5.1.4.3 The Generation Function for the Attacker

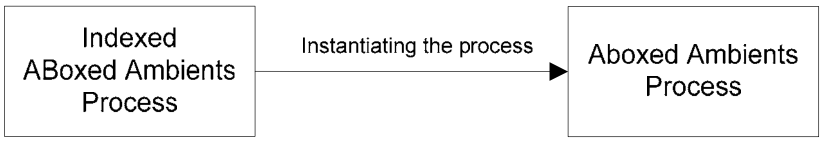
Similar to the generation function for the analysis, the generation function for the attacker takes the form $\mathcal{G}(\bullet, *)$ where $*$ is the enclosing ambient name of the attacker $\bullet$. It is defined by rewriting the component in Table 4-1 into ALFP formulae. The transformation is apparent and listed in Appendix ? for reference.

5.2 Indexed ABoxed Ambients

When validating a cryptographic protocol we often need declare multiple sites which run the same protocol in parallel in order to model the man-in-the-middle attack. This will sharply increase the effort of programming. Taking WMF protocol for example: Suppose we need program two initiators and two responders together with one server. To ensure every initiator can communication with any responder, the size of the process for initiator is not doubled but four times as big as the one of the case of single site. So is the size of responder and server program. The problem is much worse if three initiators and three responders are considered to model man-in-the-middle attack. The program size is 3×3 times as big as that of the case of single-site. As the result the effort for checking and maintaining a program becomes quite large and the reusing of the process is not easy either.

This issue motivates us to introduce indexes into a naming and consider indexes as part of the naming. Then these indexes are treated as the arguments of a process. Once the range of its values is given, the process is unfolded into its instantiated version.

This idea is similar to the principle of Meta-LySa except that we shall keep the change only within the syntax. In other word we would rather consider it as a convenient way to program a protocol and do not pretend to define semantics and thus change our analysis. All indexed processes must first be unfolded into the standard ABoxed Ambients process which is the only object language accepted by our tool. This is graphically represented as below diagram.



The extended syntax is listed below

$$\begin{aligned} P ::= \; & |_{i\in\{1,\ldots,n\}} \; P_i \\ | \; & v_{i\in\{1,\ldots,n\}} \; name_i \\ | \; & vk_{i\in\{1,\ldots,n\}} \; name_i \\ | \; & vs_{i\in\{1,\ldots,n\}} \; name_i \\ \ldots \; & \text{(others are the same as before)} \end{aligned}$$

and their equivalent relation with respect to the original ABoxed Ambients process are

$$\begin{aligned} |_{i\in\{1,\ldots,n\}} \; P_i \; & \equiv \; P_1 \mid \ldots \mid P_n \\ v_{i\in\{1,\ldots,n\}} \; name_i \; & \equiv \; (v \; name_1)\ldots(v \; name_n) \\ vk_{i\in\{1,\ldots,n\}} \; name_i \; & \equiv \; (vk \; name_1)\ldots(vk \; name_n) \\ vs_{i\in\{1,\ldots,n\}} \; name_i \; & \equiv \; (vs \; name_1)\ldots(vs \; name_n) \end{aligned}$$

We can easily generalize the idea to any number of indexes following the same ideas.

Example 5.1 Supposing there are two initiators and two responders, we rewrite WMF protocol with the indexed ABoxed Ambients as below.

$$\begin{aligned}
& (v_{i\in\{1,2\}} \; K_{AS\_i})(v_{j\in\{1,2\}} \; K_{BS\_j}) \\
& (|_{i\in\{1,2\}} \; A_i[\; |_{j\in\{1,2\}} \; (v_{i\in\{1,2\},j\in\{1,2\}} \; K_{AB\_ij}) \\
& \qquad\quad K_{AS\_i}[\text{out } A_i.\text{ in } S.\,(<A_i>^{\uparrow} \mid <B_j, K_{AB\_ij}>^{\circ}_{\mathrm{A}_{1i}} \;[\text{dest } \mathrm{S}_1])] \mid \\
& \qquad\quad (v_{i\in\{1,2\},j\in\{1,2\}} \; M_{ij}) \, K_{AB\_ij}[\text{out } A_i.\text{ in } B_j.\, <M_{ij}>^{\circ}_{\mathrm{A}_{2i}}[\text{dest } \mathrm{B}_2]] \\
& | \\
& S[\text{KeyTable} \mid \; |_{i\in\{1,2\}} \;\; |_{j\in\{1,2\}} \\
& \quad (A_i;)^{\circ}.(A_i; y_{K_{AS\_i}})^{\text{datafile}}_{\varepsilon}[\text{dest } \mathcal{D}].(B_j;\; \mathrm{y}_{K_{AB\_j}})^{\mathrm{y}_{K_{AS\_i}}}_{\mathrm{S}_1}[\text{orig } \mathrm{A}_{1i}]. \\
& \qquad\quad (B_j;\mathrm{y}_{K_{BS\_j}})^{\text{datafile}}_{\varepsilon}[\text{dest } \mathcal{D}].\mathrm{y}_{K_{BS\_j}}[\text{out } S.\text{ in } B_j. < A_i,\; \mathrm{y}_{K_{AB\_ij}} >^{\circ}_{\mathrm{S}_2}[\text{dest } \mathrm{B}_{1j}]] \\
& | \\
& |_{j\in\{1,2\}} \; B[\; |_{i\in\{1,2\}} \\
& \qquad\quad (A_i;\; \mathrm{z}_{K_{AB\_ij}})^{K_{BS\_j}}_{\mathrm{B}_{1j}}[\text{orig } \mathrm{S}_2].(;\mathrm{z})^{\mathrm{z}_{K_{AB\_ij}}}_{\mathrm{B}_{2j}}[\text{orig } \mathrm{A}_{2i}]])
\end{aligned}$$

It can be seen that the process is quite compact and readable. Especially the labels in the process are also indexed to keep the check for authentication clear. □

5.3 Summary

This chapter presented the transformations from the original analysis to the analysis that can be represented with the features available in ALFP. The correctness of these transformations can be verified by checking if the new analysis loses any estimate information compared with the original one. Based on the transformed analysis we defined the generation function which creates an ALFP formula representing the analysis of an ABoxed Ambients process.

To program a protocol and model man-in-the-middle attack in a more compact way, we little extended our syntax to profit the declaration of multi-sites. It is not part of the implementation of the analysis but defines the additional primitives and the way that how they are interpreted corresponding to the original syntax. All of these definitions are implemented by our tool in SML. The relevant code is in Appendix E for your reference.

6 Validation of Protocols

Most existing techniques for protocol validation focus on communication happening between the participants of a protocol. They assume a server can always get private keys safely. When a protocol is applied in a real network, however, this assumption is somewhat too strong. After all, a server may maintain thousands of private keys. How these keys are managed is interesting to attackers and may become the security weakness of the whole system. With the help of ambient structure, we can model key-retrieving as well as communication between participants. Thus we can remove the assumption that the private key is always safely stored and retrieved, and treat it as a part of protocol validation.

The boundary of ambients also allows us to model a private channel between two participants by assuming the location that they exist in is inaccessible to attackers. Also ambient structure enables us to allocate agents into locations other than the public-accessed network when analyzing a protocol. We can change the location of attackers and check whether the authenticity is still maintained or not during these changes.

In this chapter we first present the analysis results we have done for a number of symmetric key protocols including Wide Mouthed Frog [31], Needham-Schroeder [30], Otway-Rees [27], Yahalom [29] and Andrew Secure RPC [28]. We implement these protocols in ABoxed Ambients and present the analysis results. If any flaw is found, we shall analyze the reason and try to fix the problem. All these protocols are analyzed in a flat space of network for the convenience of comparing our analysis results with those of LySa [2].

Then we present a new attack, called chosen-protocol attack, which is reported in [25]. We show that the attacks can be detected by our tool automatically.

In the last part of this chapter, we extend our analysis into the field of hierarchical networks. First we model key-retrieving in ABoxed Ambients calculus. Then we design a series of configurations which actually are a set of assumptions about the network environment. Some protocols are supposed to execute in such configurations and validated by our analysis.

To keep our presentation clear and concise, we write processes in standard syntax specified in Chapter 2 through the presentation of this chapter. By this way our readers do not have to be disturbed by too much implementation details like canonical names, site names, indexes, etc. In the actual experiments we have taken the number of each role (except server) to be 3 in order to be sure that the man-in-the-middle attack can be modeled. The SML code of these processes available in appendix should provide such information and thus keep the completeness of the whole presentation.

6.1 Protocol Validation in Public-accessed Networks

In this section we assume (1) all participants work on the unique network and (2) keys are always safely stored and retrieved. That is for the ease of comparison between the results of LySa and those of ABoxed Ambients. We shall remove these assumptions in the later sections.

6.1.1 Validating WMF

Based on above assumption, we simplify our process for WMF by removing the process for key-retrieving as below.

$$
\begin{aligned}
&(\nu\, K_{AS})(\nu\, K_{BS})\\
&(\, A[(\nu\, K_{AB})\; K_{AS}[\text{out}\, A.\ \text{in}\, S.\ (<A>^{\uparrow} | <B,\ K_{AB}>^{\circ}_{A_1}\ [\text{dest } \mathrm{S}_1])]\ |\\
&\qquad\qquad (\nu\, M)\; K_{AB}[\text{out}\, A.\ \text{in}\, B.\ <M>^{\circ}_{A_2}\ [\text{dest } \mathrm{B}_2]]\\
&|\\
&\ S[(A;)^{\circ}.(B;\ y_{K_{AB}})^{K_{AS}}_{S_1}[\text{orig } \mathrm{A}_1].\\
&\qquad K_{BS}[\text{out}\, S.\ \text{in}\, B. <A,\ y_{K_{AB}}>^{\circ}_{S_2}\ [\text{dest } \mathrm{B}_1]]\\
&|\\
&\ B[(A;\ z_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig } \mathrm{S}_2].(;z)^{z_{K_{AB}}}_{B_2}[\text{orig } \mathrm{A}_2]])\\
&|\ \bullet
\end{aligned}
$$

We compute the least estimate that satisfies the formula $\mathcal{F}^{\mathrm{A\_DY}}_{\mathrm{RM}}$ and the location of the attackers shows that they can access only to the public-accessed network. The analysis result shows that there is no authentication violation just as the result presented in [2].

The two variants of the protocol discussed in [2] are also analyzed here: the initiator's name and the responder's name are not encrypted respectively in them (Please refer to Appendix A).

Variant 1 The specification in ABoxed Ambients for the first variant is defined as below.

$$
\begin{array}{l}
(\nu\, K_{AS})(\nu\, K_{BS}) \\
(\ A[(\nu\, K_{AB})\ K_{AS}[\text{out}\, A.\ \text{in}\, S.\ (<A>^{\uparrow} | <B,\ K_{AB}>^{\circ}_{A_1}\ [\text{dest } \mathrm{S}_1])]\ | \\
\qquad\qquad\qquad (\nu\, M)\ K_{AB}[\text{out}\, A.\ \text{in}\, B.\ <M>^{\circ}_{A_2}\ [\text{dest } \mathrm{B}_2]] \\
| \\
\ S[(A;)^{\circ}.(B;\ y_{K_{AB}})^{K_{AS}}_{S_1}[\text{orig } \mathrm{A}_1]. \\
\qquad\quad K_{BS}[\text{out}\, S.\ \text{in}\, B.(<A>^{\uparrow} | <y_{K_{AB}}>^{\circ}_{S_2}\ [\text{dest } \mathrm{B}_1])] \\
| \\
\ B[(A;)^{\circ}.\ (;z_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig } \mathrm{S}_2].(;z)^{z_{K_{AB}}}_{B_2}[\text{orig } \mathrm{A}_2]]) \\
|\ \bullet
\end{array}
$$

The $\varphi-$component of the analysis results is

$$\{(A_{2i}, B_{2j}) \mid 1 \le i, j \le n\} \cup \{(S_2, B_{1j}) \mid 1 \le j \le n\}$$

showing that static authentication fails. The pair (A_{2i}, B_{2j}) shows that some value encrypted at A_{2i} has wrongly been decrypted at B_{2j}; similarly the pair (S_2, B_{1j}) shows that a value encrypted at S_2 has been decrypted at B_{2j}. It would be interesting to compare the estimate with that of LySa presented as below

$$\{(A_{2i}, B_{2j}) \mid i \ne j, 1 \le i, j \le n\} \cup \{(\ell_{\bullet}, B_{2j}) \mid 1 \le j \le n\}$$

Here the pair $(\ell_{\bullet}, B_{2j})$ means that a value created by attackers has been decrypted at B_j.

For the pair (A_{2i}, B_{2j}) our estimate covers the result of LySa. But the pair (S_2, B_{1j}) has no overlap with the pair $(\ell_{\bullet}, B_{2j})$ and that phenomena need further investigating.

By analyzing our result, we found the pair (S_2, B_{1j}) is actually over-estimation. It is introduced because the analysis can not tell apart the order of the two inputs, $(A;)^{\circ}$ and $(;z_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig } \mathrm{S}_2]$, in ambient *B*. As the result our analysis simply keeps the set of all available messages for the second input which, in deed, should be little smaller if the effect of the first input is considered. The precision of estimate is therefore a little reduced but still safe.

After inspecting the result of LySa, we found that the pair $(\ell_{\bullet}, B_{2j})$ also belongs to the part of over-estimation. The root cause for such over-approximation is that the tree grammar used in LySa allows attackers modify the contents of an encrypted message even though they don't know the private key of the message. As this in fact strengthens the power of attackers, the approximation of LySa is still in the safe side.

The attacks illustrated in [2] can also be used to explain how the dynamic authentication fails and will not be repeated here.

Variant 2 The second variant of WMF protocol is that the responder's name is not encrypted in the message from *A* to *S*. The specification is modified as the below process.

$$
\begin{aligned}
&(\nu\, K_{AS})(\nu\, K_{BS})\\
&(\, A[(\nu\, K_{AB})\; K_{AS}[\text{out}\, A.\ \text{in}\, S.\ (<A, B>^{\uparrow} | <K_{AB}>^{\circ}_{A_1}\ [\text{dest}\ \mathrm{S}_1])]\ |\\
&\qquad\qquad (\nu\, M)\; K_{AB}[\text{out}\, A.\ \text{in}\, B.\ <M>^{\circ}_{A_2}\ [\text{dest}\ \mathrm{B}_2]]\\
&|\\
&\ S[(A, B;)^{\circ}.(;y_{K_{AB}})^{K_{AS}}_{S_1}[\text{orig}\ \mathrm{A}_1].\\
&\qquad K_{BS}[\text{out}\, S.\ \text{in}\, B.(<A>^{\uparrow} | <y_{K_{AB}}>^{\circ}_{S_2}\ [\text{dest}\ \mathrm{B}_1])]\\
&|\\
&\ B[(A;)^{\circ}.\ (;z_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig}\ \mathrm{S}_2].(;z)^{z_{K_{AB}}}_{B_2}[\text{orig}\ \mathrm{A}_2]])\\
&|\ \bullet
\end{aligned}
$$

This time the $\varphi -$ component of the analysis results becomes

$$\{(A_{2i}, B_{2j}) \,|\, 1 \le i, j \le n\} \cup \{(A_{1i}, S_1) \,|\, 1 \le i \le n\}$$

Here the pair (A_{1i}, S_1) is over-estimate. This is observed at the two continuous inputs, $(A, B;)^{\circ}$ and $(;\ y_{K_{AB}})^{K_{AS}}_{S_1}[\text{orig}\ \mathrm{A}_1]$ in ambient *S*, where the estimate of the second input does not consider the existing of its previous one. In LySa, the reported result is

$$
\begin{aligned}
&\{(A_{2i}, B_{2j}) \,|\, 1 \le i, j \le n\} \cup\\
&\{(A_{1i}, \ell_{\bullet}) \,|\, 1 \le i \le n\} \cup \{(\ell_{\bullet}, B_{1j}) \,|\, 1 \le j \le n\}
\end{aligned}
$$

For the similar reason presented in the variant 1, the last two pairs are over-approximation.

The comparison of the estimate for WMF between LySa and ABoxed Ambients shows that both of them are safe and correct. The over-approximations introduced by these two analyses, however, are different depending on the properties of the estimate of their analysis specifications.

6.1.2 Validating Needham-Schroeder

Following the approach we applied when programming WMF protocol in Chapter 2, we first expand the protocol narration to bridge the gap between informal and formal specification as below.

1. $A \to\ :\ A,\ B,\ R_A$ {assuming R_A is a new value}
   $\to S\ :\ y_A,\ y_B,\ y_{R_A}$ {check $y_A = A, y_B = B$}
2. $S \to\ :\ K_{AS}[y_A, y_B, K_{AB}, K_{BS}[y_A, K_{AB}]$[dest B]][dest A] {assuming K_{AB} is a new key}
   $\to A\ :\ x_1$
   $A\ :$ decrypt x_1 as $K_{AS}[x_{R_A}, x_B, x_{K_{AB}}, x_{K_{BS}[A,K_{AB}]}]$[orig S] {check $x_{R_A} = R_A, x_B = B$}
3. $A \to\ :\ x_{K_{BS}[A,K_{AB}]}$
   $\to B\ :\ z_1$
   $B\ :$ decrypt z_1 as $K_{BS}[z_A, z_{K_{AB}}]$[orig S] {check $z_A = A$}
4. $B \to\ :\ z_{K_{AB}}[R_B]$[dest A]
   $\to A\ :\ x_2$
   $A\ :$ decrypt x_2 as $x_{K_{AB}}[x_{R_B}]$[orig B]
5. $A \to\ :\ x_{K_{AB}}[x_{R_B} + 1]$[dest B]
   $\to B\ :\ z_2$
   $B\ :$ decrypt z_2 as $z_{K_{AB}}[z_{R_B+1}]$[orig A] {check $z_{R_B+1} = R_B + 1$}
6. $A \to\ :\ x_{K_{AB}}[M]$[dest B] {assuming is M a new value}
   $\to B\ :\ z_3$
   $B\ :$ decrypt z_3 as $z_{K_{AB}}[z_M]$[orig A]

As presented in the above extended narration, we need check the successor of the random number R_B at the third line of the step 5. Since successor operation has not been provided in ABoxed Ambients, we can not encode it directly but have to find an alternative way which can capture the characteristic of the operator. The characteristic of the successor operation is identified and could be summarized as below:

- If the attacker gets the random number R_B then he has knowledge of $R_B + 1$.

Following the participant we encode $R_B + 1$ as the pair (R_B, R_B). The arity of the message including $R_B + 1$ is correspondingly increased one. If the attacker for some reason knows R_B, then he can create the message including two R_B s.

Following the extended protocol narration the Needham-schroeder protocol in ABoxed Ambients is then programmed as the following process.

$$
\begin{array}{l}
(\nu\, K_{AS})(\nu\, K_{BS}) \\
(A[(\nu\, R_A)\; p[\text{out } A.\text{ in } S.\; < A,\, B,\, R_A >^{\uparrow}]\; | \\
\quad (R_A; x_{K_{AB}})_{A_1}^{K_{AS}}[\text{orig } S_1]. \\
\quad (x_{R_B})_{A_2}^{x_{K_{AB}}}[\text{orig } B_2].x_{K_{AB}}[\text{out } A.\text{ in } B.\; < x_{R_B}, x_{R_B} >_{A_3}^{\circ}[\text{dest } B_3]]\; | \\
\quad (\nu\, M)x_{K_{AB}}[\text{out } A.\text{ in } B.\; <M>_{A_4}^{\circ}[\text{dest } B_4]]\;)] \\
| \\
S[(A,\, B;\, y_{R_A})^{\circ}.(\nu\, K_{AB}) \\
\quad K_{AS}[\text{out } S.\text{ in } A.\;(< y_{R_A},\, K_{AB} >_{S_1}^{\circ}[\text{dest } A_1]\; | \\
\qquad K_{BS}[\text{out } K_{AS}.\text{out } A.\text{in } B. < A, K_{AB} >_{S_2}^{\circ}[\text{dest } B_1]])]] \\
| \\
B[(A; z_{K_{AB}})_{B_1}^{K_{BS}}[\text{orig } S_2].(\nu\, R_B)\;(z_{K_{AB}}[\text{out } B.\text{ in } A. < R_B >_{B_2}^{\circ}[\text{dest } A_2]\;]| \\
\quad (R_B, R_B;)_{B_3}^{z_{K_{AB}}}[\text{orig } A_3].(z_M)_{B_4}^{z_{K_{AB}}}[\text{orig } A_4].\cdots z_M \cdots)]) \\
|\; \bullet
\end{array}
$$

The analysis result is summarized as

$$
\begin{array}{l}
\{(S_1, A_{1i}) \mid 1 \le i \le n\} \cup \{(S_1, B_{1j}) \mid 1 \le j \le n\} \cup \{(B_{2j}, A_{2i}) \mid i \ne j, 1 \le i \le n\} \\
\{(A_{3i}, B_{3j}) \mid i \ne j, 1 \le i \le n\} \cup \{(A_{4i}, B_{4j}) \mid i \ne j, 1 \le j \le n\} \cup \\
\{(A_{4i}, A_{2i}) \mid 1 \le i \le n\} \cup \{(B_{2j}, B_{4j}) \mid 1 \le j \le n\}
\end{array}
$$

By inspecting the results, we found that the messages sent by some participants are received by themselves or others unexpectedly. The flaw is also reported by LySa.

Correcting the flaw The fix for the flaw suggested in [2] is to add extra components $(u_1, u_2, \cdots)$ in the encrypted messages. The extended narration for the corrected steps is presented below.

$$
\begin{array}{lll}
4.\; B \rightarrow & : z_{K_{AB}}[u_1, R_B][\text{dest A}] & \\
\quad \rightarrow A & : x_2 & \\
\quad A & : \text{decrypt } x_2 \text{ as } x_{K_{AB}}[x_{u_1}, x_{R_B}][\text{orig B}] & \{\text{check } x_{u_1} = u_1\} \\
5.\; A \rightarrow & : x_{K_{AB}}[u_2, x_{R_B} + 1][\text{dest B}] & \\
\quad \rightarrow B & : z_2 & \\
\quad B & : \text{decrypt z}_2 \text{ as } z_{K_{AB}}[x_{u_2}, z_{R_B+1}][\text{orig A}] & \{\text{check } x_{u_2} = u_2, z_{R_B+1} = R_B + 1\}
\end{array}
$$

Our experiment shows that the change fixes the problem and the $\varphi-$component is empty now.

6.1.3 Validating Other Protocols

Similar to the experiment on Needham-Schroeder, we also validate other symmetric key protocols, including Otway-Rees, Yahalom and Andrew Secure RPC. For every protocol we give its extended narration and the corresponding process in ABoxed Ambients in Appendix B. Also their analysis results are discussed and flaws (if any) are corrected.

6.2 Chosen Protocol Attack

A protocol may be completely secure alone, but may become insecure when another protocol exists that can be carried out with the same key pair. This idea has been shown to be possible in [25] which discussed how protocol interactions can weaken the security of one or both protocols. Accordingly a new attack, the chosen-protocol attack, is defined: A new protocol is designed to interact with an existing protocol to create a security hole.

In this section we analyze one example of the chosen-protocol attack presented in [25] and show our analysis can detect this kind of attack besides the classical attacks,such as replay attack, modification, man-in-the-middle, and so on.

In the chosen-protocol attack, there are one target protocol and one chosen protocol. In our case, the target protocol is the WMF protocol and the chosen protocol is Secure Login protocol. The WMF protocol is just as before and specifically the private key length is 192-bit which is used in the example of [25]. We here give a brief introduction to the Secure Login protocol.

Secure Login Protocol proceeds as below steps:

1. Mallory sends to Alice:

   $$M_0 = LoginChallenge, N$$

   where N is a 64-bit random number.

2. Alice responds by sending the server S:

   $$M_1 = LoginRequest, A, K_{AS}[N, hash(passphrase), M]$$

   where *hash(passphrase)* is the hashed login password of Alice for logging on Mallory, and A and M are IDs for Mallory and Alice individually.

3. Server S verifies the IDs and then sends to Mallory:

   $$M_2 = LoginMessage, K_{MS}[A, hash(N, hash(passphrase))]$$

4. Mallory verifies the hash last.

Please note that only Alice and the server S know the original *passphrase* of Alice, and Mallory just knows *hash(passphrase)* and N in the protocol.

We summarize the two protocols in Table 6-1.

$1. A \to S: A, K_{AS}[B, K_{AB}]$	$1. M \to A: LoginChallenge, N$
$2. S \to B: K_{BS}[A, K_{AB}]$	$2. A \to S: LoginRequest, A, K_{AS}[N, hash(passphrase), M]$
$3. A \to B: K_{AB}[M]$	$3. S \to M: LoginMessage, K_{MS}[hash(N, hash(passphrase)), A]$

Table 6-1: WMF protocol and Secure Login protocol

The chosen-protocol attack then conducts as the following steps:

1. Mallory sends *B* as *N* in the first step of Secure Login protocol.
2. Mallory eavesdrops on and catches Alice's response to the server *S*, removes the *LoginRequest* header and forwards the rest of the message to the server *S* as the first step of WMF protocol.
3. *S* thinks the request is valid for a secure session with Bob from Alice. So he sends the message to Bob.
4. Now Mallory can impersonate Alice to send messages to Bob. Bob is convinced.

We aim at modeling the protocol interactions and detect the security hole with our analysis. First, Mallory, a malicious legal site, is treated as an attacker which knows *hash(passphrase)*, its own ID *M* and the private key K_{MS}. We model the actions of Alice, the server *S* and Bob. The interactions of the two protocols are modeled by combining the activities of the same role in the two protocols. Based on the activities of WMF protocol, for instance, Alice should have following activities of Secure Login protocol:

- be able to receive the message *LoginChallenge, N*;
- send $LoginRequest, A, K_{AS}[N, hash(passphrase), M]$ to *S*.

In [25], the length of the pair *hash(passphrase), M* together is equal to that of K_{AB}. So *hash(passphrase)* with *M* can be thought as one private key K_{AB}. To capture this we use one name *hash(passphrase)_M* to represent the pair. By this way, the arities of the two messages encrypted by K_{AS} are equal and the interactions of the protocols become possible. As Mallory knows both *hash(passphrase)* and *M*, he certainly have the knowledge *hash(passphrase)_M* also.

Similarly we extend the activities of the server *S* of WMF protocol as below.

- Receiving the message $LoginRequest, A, K_{AS}[N, hash(passphrase), M]$ from Alice.

In the step 3 of Secure Login protocol, the action of sending message from *S* to Mallory is not so interesting since these messages are sent to the attacker and their contents are supposed to be accessible to the attacker.

Observing that the headers of the messages are not part of the interactions, we remove them from the messages and the activities of Alice, the server *S* and Bob are programmed as the below process (we underline the added sub-processes compared with the original process of WMF).

$$\begin{array}{l}
(\nu\ hash\_pwa\_M)(\\
(\nu\ K_{AS})(\nu\ K_{BS}) \\
(\ A[(\nu\ K_{AB})\ K_{AS}[\text{out } A.\ \text{in } S.\ (<A>^{\uparrow} | <B,\ K_{AB}>^{\circ}_{A_1}\ [\text{dest S}_1])]\ | \\
\qquad (\nu\ M)\ K_{AB}[\text{out } A.\ \text{in } B.\ <M>^{\circ}_{A_2}\ [\text{dest B}_2]\ | \\
\qquad \underline{(;x_N)^{\circ}.\ K_{AS}[\text{out } A.\ \text{in } S.\ (<A>^{\uparrow} | <x_N, hash\_pwa\_M>^{\circ}_{A_3}\ [\text{dest S}_3])]]} \\
| \\
S[(A;)^{\circ}.(B;\ y_{K_{AB}})^{K_{AS}}_{S_1}[\text{orig A}_1]. \\
\qquad K_{BS}[\text{out } S.\ \text{in } y_B.<A,\ y_{K_{AB}}>^{\circ}_{S_2}\ [\text{dest B}_1]\ | \\
\qquad \underline{(A;)^{\circ}.(;y_n, y_{hash\_pwa\_M})^{K_{AS}}_{S_3}[\text{orig A}_3]]} \\
| \\
B[(A;\ z_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig S}_2].(;x)^{z_{K_{AB}}}_{B_2}[\text{orig A}_2]]) \\
|\ \bullet)
\end{array}$$

The experiment result is

$$\begin{array}{l}
\{(A_{3i}, S_1) \mid 1 \le i \le n\} \cup \\
\{(A_{1i}, S_3) \mid 1 \le i \le n\} \cup \{(\ell_\bullet, B_{2j}) \mid 1 \le j \le n\}
\end{array}$$

As expected, the estimate $(\ell_\bullet, B_{2j})$ confirms the flaws pointed by [25] by showing that attacker-composed messages are successfully decrypted by Bob who, however, think the message is from Alice. In particular, the pairs (A_{3i}, S_1) and (A_{1i}, S_3) provide more useful information about the problem itself. They tell us the messages encrypted at A_3 and A_1 are decrypted at S_1 and S_3 wrongly respectively. By reviewing their intended crypto-points, we found the expected intention for A_1 and A_3 is S_1 and S_3. Since the activities at the points (A_1, S_1) and (A_3, S_3) belong to the object protocol and chosen protocol individually, the estimate clearly shows there are unexpected interactions between two protocols. Following this clue, we found the root cause of the problem is that the server S uses the same key K_{AS} for both two protocols. Therefore the correction of the flaw is to let Secure Login protocol use another private key K'_{AS} . The experiment on the modified protocol then shows the static authentication success on the corrected version.

6.3 Protocol Validation in Hierarchical Networks

From this section we turn to the field of hierarchical networks. We first remove the assumption about key-retrieving on authentication servers and then consider it as part of protocol validation: If any key can be gotten by unauthorized participants or attackers, our analysis should find the problem and report that the protocol is insecure. We here model two cases: (1) All private keys are stored in a data file; (2) A special database server is allocated for the services of key-storing and key-querying. The first case gives rise to a hierarchical structure between an authentication server and a data file. In the second case, an authentication server and a database server are first arranged in a flat space. But our analysis shows that to ensure authenticity a local network should be introduced to provide a private channel between the servers. As the result we form a hierarchical network in which the servers are nested in the local network that is further embedded in the public-accessed network.

In a hierarchical space, both agents and attackers have more choices on locations. For agents this means that they may work at different network environment other than the Internet. For example, we can model the change of locations by specifying separate processes for every possible location although a site is not movable in our calculus. For attackers this implies that they may access to some local networks besides the Internet so that they may access to some private channels. Where participants and attackers may appear depends on the structure of the analyzed network and our assumptions. In this section we shall construct some example networks and experiment with them by adjusting the position of participants or attackers in these networks.

We restrict our attention to the estimates that satisfy the formula $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$. Therefore, usually a cryptographic protocol is also secure in hierarchical networks if it is secure in the unique network, the Internet. This is because the boundary of local network actually protects the communication between agents inside the local network from the eavesdropping of the attacker outside. Even though we can assume the attacker is able to access to any local networks, they just acquire their capability as before in the flat space of network. On the other hand, we could take advantage of the boundary of local networks to improve the performance of a protocol by removing some encryptions. Under proper assumptions about participants and attackers, we illustrate how our analysis assists to the optimization of a protocol implemented in a hierarchical network.

We chose WMF protocol and its variants as our target protocols because the protocol is simple as well as typical. It has all roles used in most cryptographic protocols, including authentication server(s), initiators and responders. Also the authentication server need acquire a mass of keys for the initiators and responders. The presentation in this section is organized as a serial of configurations in which assumptions for every participant of the protocol and the attacker are claimed at first. Then the protocols are validated under the assumed configuration. Last the estimate is summarized and analyzed.

6.3.1 Modeling the Key-retrieving in Hierarchical Networks

Configuration 1: Private keys are stored in a file on the authentication server.

(1) All participants are on the Internet;

(2) The key file is located in the authentication server; attackers have no knowledge about the file, i.e. file name, but they know the name of all participants of the protocol;

(3) Attackers can access to the Internet only.

Test 1

The idea is to use an ambient to represent a data file inside the server ambient which reads key values with the parent-child input. The corresponding process has been programmed in Table 2-9, Chapter 2. We here graphically present its ambient structure together with attackers in Figure 6-1.

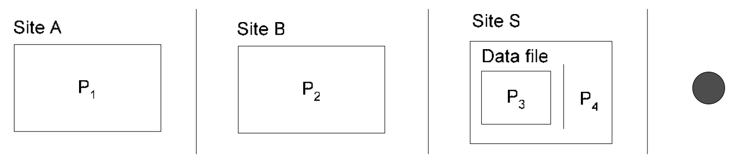


Figure 6-1: Ambient structure of Configuration 1

The experiment shows the protocol is safely applied under above structure and assumptions.

One, however, may suggest that attackers could show up inside the server. But the location bounded by the boundary of the server *S* is actually not a network location where attackers may exist. We here think that the process is modeling the computer (CPU) reading a file from its local hard disk. The communication between them usually can not be disturbed by any network attackers.

Configuration 2: Private keys are stored on a database server (which is on the Internet).

(1) All participants are on the Internet;

(2) Attackers' initial knowledge includes the name of all participants and the name of the database server;

(3) Attackers can access to the Internet only.

We present the structure of the ambients as below figure according to our assumptions.

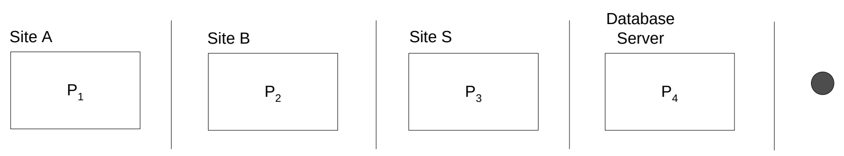


Figure 6-2: Ambient structure of Configuration 2

Test 1

The communication between the authentication server and the database server is through the packet now. For example, if *S* needs the private key of *A*, we can describe the process as below extended narration:

$$\begin{array}{ll}
1. & S \to \ : A \\
1'. & \to DBS : y_A \qquad \{\text{check } y_A = A\} \\
2. & DBS \to : y_A, K_{AS} \\
2'. & \to S \ : x_A, x_{K_{AS}} \qquad \{\text{check } x_A = A\}
\end{array}$$

The first line describes that *S* asks the private key of *A* from *DBS* while line 1′ checks the ID of *A*. Then *DBS* replies the message including *A*’s ID and its private key in the second step. Last *S* checks the ID to determine which participant the private key is for. We can similarly apply the method for reading other keys stored on the authentication server. We modify our process of WMF protocol as below and the sub-processes for communication between *S* and *DBS* are underlined.

$$\begin{array}{l}
(\nu\ K_{AS})(\nu\ K_{BS}) \\
(A[(\nu\ K_{AB})\ K_{AS}[\text{out } A.\text{ in } S.\ (<A>^{\uparrow} \mid < B, K_{AB} >^{\circ}_{\mathrm{A}_1} [\text{dest } \mathrm{S}_1])] \mid \\
\qquad\qquad (\nu\ M)\ K_{AB}[\text{out } A.\text{ in } B.\ <M>^{\circ}_{\mathrm{A}_2} [\text{dest } \mathrm{B}_2]] \\
\mid \\
B[(A;\ \mathrm{z}_{K_{AB}})^{K_{BS}}_{\mathrm{B}_1} [\text{orig } \mathrm{S}_2].(;z)^{\mathrm{z}_{K_{AB}}}_{\mathrm{B}_2} [\text{orig } \mathrm{A}_2]] \\
\mid \\
S[\ (A;)^{\circ}.(\underline{p[\text{out } S.\text{in } DBS.<A>^{\uparrow}]} \mid \underline{(A; y_{K_{AS}})}^{\circ}.(B;\ \mathrm{y}_{K_{AB}})^{\mathrm{y}_{K_{AS}}}_{\mathrm{S}_1} [\text{orig } \mathrm{A}_1]. \\
\qquad (\underline{p[\text{out } S.\text{in } DBS.<B>^{\uparrow}]} \mid \underline{(B;\mathrm{y}_{K_{BS}})}^{\circ}.\mathrm{y}_{K_{BS}}[\text{out } S.\text{ in } B.< A,\ \mathrm{y}_{K_{AB}} >^{\circ}_{\mathrm{S}_2} [\text{dest } \mathrm{B}_1]]))] \\
\mid \\
DBS[!\underline{(A;)}^{\circ}.\underline{p[\text{out } DBS.\text{in } S.<A, K_{AS}>^{\uparrow}]} \mid \\
\qquad !\underline{(B;)}^{\circ}.\underline{p[\text{out } DBS.\text{in } S.<B, K_{BS}>^{\uparrow}]}] \\
\mid \bullet)
\end{array}$$

Table 6-2: Process for Test 1 of Configuration 2

As the process presents, the messages transferred between *S* and *DBS* are public-accessed. This could leave security flaws for the attacker. Our analysis also shows that attackers’ knowledge includes K_{AS}, K_{BS} and K_{AB}, and thus the protocol are totally cracked.

One choice to fix the problem could be to encrypt the messages containing private keys with another private key shared by *S* and *DBS*. However it is not very secure to encrypt all these messages with the same key since all these messages may be cached by attackers to analyze the key.

In the real life, authentication server and database server are not located in the Internet directly but are often put in a local network for their importance and expensive value. Suppose attackers can not access to a local network, then the local network serves as a private channel between *S* and *DBS*. That idea is presented in the next Configuration.

Configuration 3**:** Private keys are stored on a database server which is in a local network.

(1) Initiators and responders are on the Internet; the authentication server and database server are located in the office network;
(2) Attackers' initial knowledge includes the name of all participants, the name of the office network and the name of the database server;
(3) Attackers can access to the Internet only.

We present the structure of the ambients as below figure according to our assumptions.

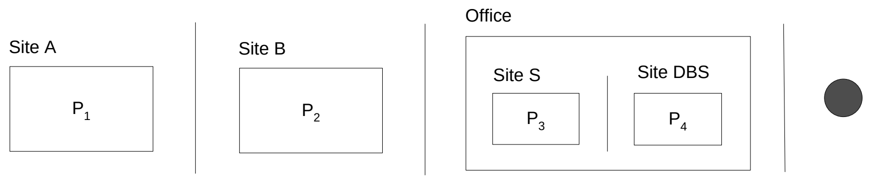


Figure 6-3: Ambient structure of Configuration 3

Test 1

In the first test of this configuration, we try to keep the original processes for the ambients *S* and *DBS*. The changes are then focused on the structure of ambients and mobility primitives which direct the move of packets. The process is declared in Table 6-3 where the changes compared with the process in Table 6-2 are underlined.

$$
\begin{array}{ll}
(\nu\ K_{AS})(\nu\ K_{BS}) & \\
(A[(\nu\ K_{AB})\ K_{AS}[\text{out } A.\underline{\text{in } OFF}.\ \text{in } S.\ (<A>^{\uparrow} \mid < B, K_{AB} >^{\circ}_{A_1}\ [\text{dest } \mathrm{S}_1])] \mid & \\
\qquad\qquad (\nu\ M)\ K_{AB}[\text{out } A.\ \text{in } B.\ <M>^{\circ}_{A_2}[\text{dest } \mathrm{B}_2]] & \\
\mid & \\
B[(A;\ \mathrm{z}_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig } \mathrm{S}_2].(;z)^{z_{K_{AB}}}_{B_2}[\text{orig } \mathrm{A}_2]] & \\
\mid & \\
\underline{OFF}[S[\ (A;)^{\circ}.(p[\text{out } S.\text{in } DBS.<A>^{\uparrow}] \mid & \\
\qquad (A; y_{K_{AS}})^{\circ}.(B;\ \mathrm{y}_{K_{AB}})^{y_{K_{AS}}}_{S_1}[\text{orig } \mathrm{A}_1]. & (1) \\
\qquad (p[\text{out } S.\text{in } DBS.<B>^{\uparrow}] \mid & \\
\qquad (B;\mathrm{y}_{K_{BS}})^{\circ}.\mathrm{y}_{K_{BS}}[\text{out } S.\ \underline{\text{out } OFF}. & (2) \\
\qquad\qquad\qquad \text{in } B. < A,\ \mathrm{y}_{K_{AB}} >^{\circ}_{S_2}[\text{dest } \mathrm{B}_1]]))] & \\
\quad \mid & \\
\quad DBS[!(A;)^{\circ}.p[\text{out } DBS.\text{in } S.<A, K_{AS}>^{\uparrow}] \mid & \\
\qquad !(B;)^{\circ}.p[\text{out } DBS.\text{in } S.<B, K_{BS}>^{\uparrow}]]] & \\
\mid \bullet) &
\end{array}
$$

Table 6-3: Process for Test 1 of Configuration 3

As the attacker is on the Internet, he can not capture the packet p and thus there should be no way for him to acquire the private key K_{AS} and K_{BS}. But he can still deliver messages into the ambients S and DBS as he know the names of the authentication server and database server. The actual experiment, however, shows that attackers know K_{BS}, i.e. $K_{BS} \in \rho(z_\bullet)$. The $\varphi-$component is quite long and listed below

$$
\begin{aligned}
&\{(A_{2i}, \ell_\bullet) \mid 1 \le i \le n\} \cup \{(S_2, \ell_\bullet)\} \cup \{(\ell_\bullet, S_1)\} \\
&\cup \{(\ell_\bullet, B_{1j}) \mid 1 \le j \le n\} \cup \{(\ell_\bullet, B_{2j}) \mid 1 \le j \le n\}
\end{aligned}
$$

We are interested in why the estimate counters our intuitive. First we sense that there are two points (their lines are marked with (1) and (2) in Table 6-3) which the server can unexpectedly receive messages from attackers. For example, if attackers send S a message A, $n_\bullet$ where $n_\bullet \in \rho(z_\bullet)$, then it can be received by S at the line of the sub-process marked with (1). This can be graphically illustrated as

$$
\overset{S}{\boxed{< A, n_\bullet >^{\circ} \ \mid\ (A; y_{K_{AS}})^{\circ}.P}} \ \rightarrow\ \overset{S}{\boxed{P\{y_{K_{AS}} \rightarrow n_\bullet\}}}
$$

This gives that $\rho(z_\bullet) \subseteq \rho(y_{K_{AS}})$. We can similarly analyze the second point and get the result $\rho(z_\bullet) \subseteq \rho(y_{K_{BS}})$. Both the above conclusions are also confirmed by the automated analysis. Attackers may also deliver messages to *DBS*. But they only cause *DBS* to send more packet *p* to *S* that in fact has no side-effect for authenticity.

The analysis results, on the other hand, tell us that $K_{BS} \in \rho(y_{K_{AB}})$. Since this is the only way K_{BS} can be sent onto the Internet, we can immediately sense that reconstructing how it happens is very important for identifying whether or not $K_{BS} \in \rho(z_{\bullet})$ is an over-estimate. After inspecting the possible values for the variables $y_{K_{AS}}$ and $y_{K_{BS}}$, we can simulate how analysis computes the estimate as below steps:

1. In the analysis the mailbox of DBS is initially

   $$\overset{DBS}{\boxed{< A, K_{AS} >^{\circ} \mid <B; K_{BS} >^{\circ}}}$$

   by the over-approximation of the child-to-parent output from the packet *p* to the enclosing ambient *DBS*.

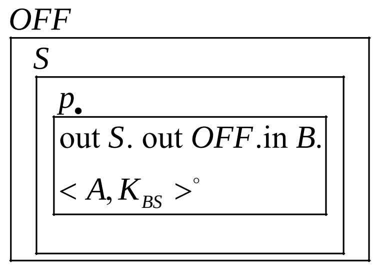
2. Suppose both $y_{K_{AS}}$ and $y_{K_{BS}}$ are the name *DBS* which is given by attackers' message. The continuous processes at the line (1) and (2) in Table 6-3 give the following structure of *S* and its evolvement

   $$\overset{S}{\boxed{\overset{DBS}{\boxed{\begin{array}{l}< A, K_{AS} >^{\circ} \mid \\ < B, K_{BS} >^{\circ}\end{array}}} \mid (B; y_{K_{AB}})^{DBS}.P}} \rightarrow \overset{S}{\boxed{\overset{DBS}{\boxed{< A, K_{AS} >^{\circ}}} \mid P\{y_{K_{AB}} \rightarrow K_{BS}\}}}$$

   This illustrates the reason for $K_{BS} \in \rho(y_{K_{AB}})$. Please note that we are simulating the computation of analysis: The ambient *DBS* constructed inside *S* is treated just as the same one outside *S*. Therefore their mailboxes are merged together.

   As only DBS' message contains the key K_{BS}, we are sure that above steps is the *only* way used by analysis to achieve the estimate $K_{BS} \in \rho(y_{K_{AB}})$.

3. Now suppose $y_{K_{BS}}$ at the line (2) in Table 6-3 is any packet name known by attacker, i.e. $p_{\bullet}$, the continuous process gives an estimate of the ambient in the form



As the figure shows, the mobility primitives will direct the packet to move onto the Internet. Then attackers can catch it and eventually get the key K_{BS} as $p_\bullet \in \rho(z_\bullet)$.

Our assumption is possible even though in the second step we have chosen $y_{K_{BS}}$ to be *DBS* while now we assign $y_{K_{BS}}$ the value $p_\bullet$. By reviewing the scratch of the process

$$(B;\mathrm{y}_{K_{BS}})^\circ.\mathrm{y}_{K_{BS}}[...]$$

we know that the analysis will attempt to simulate the creation of all ambients with the name that $y_{K_{BS}}$ may be bound to.

Summarizing the above steps, we are sure that the estimate $K_{BS} \in \rho(z_\bullet)$ is an over-approximation given by the estimates of the first and third steps. Accordingly we can remove the pair $(\ell_\bullet, B_{1j})$ for the $\varphi-$ component since K_{BS} is needed to decrypt message at the point of B_{1j}.

But the other pairs are still useful and can be illustrated by the attacks listed below.

$M_A \to S:\ A,\ K_\bullet$	$A \to S\ :\ A, K_{AS}[B, K_{AB}]$
$S \to B\ :\ K_{BS}[A,\ K_\bullet]$	$S \to B\ :\ K_{BS}[A, K_{AB}]$
$M_A \to B:\ K_\bullet[m_\bullet]$	$M \to S\ :\ B, K_\bullet$
	$S \to M_B:\ K_\bullet[A, K_{AB}]$
	$A \to B\ :\ K_{AB}[m]$
Attack 1	Attack 2

In attack 1, *B* finally believes that he is communicating with *A* although he is communicating with the attacker. The second attack makes use of the security hole at the line (2) in Table 6-3 by cheating *S* to send the attacker the key K_{AB} and accordingly the attacker can read any message sent from *A* to *B*.

Test 2

In the first test we introduced the local network *OFF* and showed the keys K_{AS} and K_{BS} are secure stored and retrieved. But there are still other security flaws existing in the protocol. The root cause for these security issues is that the authentication server can not distinguish the packets from the database server with the packets from attackers. In this sub-section we try to fix the problem with a private key K_{S_DB} only shared between the authentication server and the database server. Since now attackers can not cache packets communicated between the two servers which are both located in the office network, we do not have to worry about that the key may be explored by attackers.

The process is then declared in Table 6-4 on the next page and the major changes compared with the process in Table 6-3 are underlined.

$$(\nu\ K_{AS})(\nu\ K_{BS})$$

$$(A[(\nu\ K_{AB})\ K_{AS}[\text{out } A.\text{in } OFF.\ \text{in } S.\ (<\!A\!>^{\uparrow} | <B, K_{AB}>^{\circ}_{A_1}\ [\text{dest } \mathrm{S}_1])] \,|$$

$$(\nu\ M)\ K_{AB}[\text{out } A.\ \text{in } B.\ <\!M\!>^{\circ}_{A_2}[\text{dest } \mathrm{B}_2]]$$

$$|$$

$$B[(A;\ z_{K_{AB}})^{K_{BS}}_{B_1}[\text{orig } \mathrm{S}_2].(;z)^{z_{K_{AB}}}_{B_2}[\text{orig } \mathrm{A}_2]]$$

$$|$$

$$(\nu\ K_{S\_DB})$$

$$OFF[S[\ (A;)^{\circ}.(p[\text{out } S.\text{in } DBS.<\!A\!>^{\uparrow}]\,|$$

$$\underline{(A; y_{K_{AS}})^{K_{S\_DB}}_{SD_1}}[\text{orig } \mathrm{D}_1].(B;\ y_{K_{AB}})^{y_{K_{AS}}}_{S_1}[\text{orig } \mathrm{A}_1].$$

$$(p[\text{out } S.\text{in } DBS.<\!B\!>^{\uparrow}]\,|$$

$$\underline{(B; y_{K_{BS}})^{K_{S\_DB}}_{SD_2}}[\text{orig } \mathrm{D}_2].y_{K_{BS}}[\text{out } S.\ \text{out } OFF.\ \text{in } B. < A,\ y_{K_{AB}} >^{\circ}_{S_2}[\text{dest } \mathrm{B}_1]]))]$$

$$|$$

$$DBS[!(A;)^{\circ}.K_{S\_DB}[\text{out } DBS.\text{in } S.\underline{<A, K_{AS}>^{\circ}_{D_1}}[\text{dest } \mathrm{SD}_1]]\,|$$

$$!(B;)^{\circ}.K_{S\_DB}[\text{out } DBS.\text{in } S.\underline{<B, K_{BS}>^{\circ}_{D_2}}[\text{dest } \mathrm{SD}_2]]]]$$

$$|\,\bullet)$$

Table 6-4: Process for Test 2 of Configuration 3

Comparing with the process in Table 6-3, we use the parent-child input instead of the local input in *S* and symmetrically replace the child-to-parent output with the local output in *DBS*. We also add labels to declare the crypto-points and authentication origins or intentions to compliant with the definition of the syntax.

The estimate of the process shows that there is no security flaw in the protocol.

Test 3

When there are thousands of keys managed by the database server, one can imagine that encrypting and decrypting every key required by the authentication server are very time-consuming. To fix this problem, we here suggest another way for the authentication server to determine whether or not a packet is from the database server.

The idea is we use a unique number shared between the two servers but not known by others. This idea can be illustrated as:

$$\begin{array}{ll}
1.\ \ S \rightarrow\ \ : A & \\
1'. \rightarrow DBS: y_A & \{\text{check } y_A = A\} \\
2.\ \ DBS \rightarrow: u_1, y_A, K_{AS} & \\
2'.\ \ \rightarrow S\ \ : x_{u_1}, x_A, x_{K_{AS}} & \{\text{check } x_{u_1} = u_1 \text{ and } x_A = A\}
\end{array}$$

The first line describes that *S* asks the private key of *A* from *DBS* while line 1′ checks the ID of *A*. This time *DBS* replies the message including *A*'s ID and its private key together with the unique number u_1. By checking the unique number *S* ensures that the message is from DBS and believes the key received is not compromised. We can similarly apply the method for reading other keys stored on the authentication server. The process is then declared in Table 6-5 and the changes compared with the process in Table 6-3 are underlined.

$$\begin{array}{l}
(v\ K_{AS})(v\ K_{BS}) \\
(A[(v\ K_{AB})\ K_{AS}[\text{out } A. \text{in } OFF.\ \text{in } S.\ (<A>^{\uparrow} \mid < B, K_{AB} >^{\circ}_{A_1}\ [\text{dest } \mathrm{S}_1])] \mid \\
\qquad\qquad (v\ m)\ K_{AB}[\text{out } A.\ \text{in } B.\ <m>^{\circ}_{A_2} [\text{dest } \mathrm{B}_2]] \\
\mid \\
B[(A;\ \mathrm{z}_{K_{AB}})^{K_{BS}}_{B_1} [\text{orig } \mathrm{S}_2].(;z)^{z_{K_{AB}}}_{B_2} [\text{orig } \mathrm{A}_2]] \\
\mid \\
(v\ u_1)(v\ u_2) \\
OFF[S[\ (A;)^{\circ}.(p[\text{out } S. \text{in } DBS.<A>^{\uparrow}] \mid \\
\qquad \underline{(u_1, A; y_{K_{AS}})}^{\circ}.(B;\ \mathrm{y}_{K_{AB}})^{y_{K_{AS}}}_{S_1} [\text{orig } \mathrm{A}_1]. \\
\qquad (p[\text{out } S. \text{in } DBS.<B>^{\uparrow}] \mid \\
\qquad \underline{(u_2, B; \mathrm{y}_{K_{BS}})}^{\circ}.\mathrm{y}_{K_{BS}} [\text{out } S.\ \text{out } OFF.\ \text{in } B. < A,\ \mathrm{y}_{K_{AB}} >^{\circ}_{S_2} [\text{dest } \mathrm{B}_1]]))] \\
\quad \mid \\
\quad DBS[!(A;)^{\circ}.p[\text{out } DBS. \text{in } S.\underline{<u_1, A, K_{AS}>}^{\uparrow}] \mid \\
\qquad !(B;)^{\circ}.p[\text{out } DBS. \text{in } S.\underline{<u_2, B, K_{BS}>}^{\uparrow}]]] \\
\mid \bullet)
\end{array}$$

Table 6-5: Process for Test 3 of Configuration 3

Our experiment result shows no authentication is violated in this solution and more important we do not encrypt or decrypt messages sent by *DBS* any more.

6.3.2 Experimenting with WMF Protocol in Hierarchical Networks

In the last sub-section, we used our analysis to verify all kinds of suggestions for key-retrieving. Our analysis indeed functions as a tool which judges weather or not a solution is secure and detects any possible flaw of the solution. In this sub-section we continuously present how the analysis is useful in validating WMF protocol or its variants (we shall call them WMF-V1 and WMF-V2 for the variant one and two respectively) under our assumed configurations. Through this sub-section, we use the solution presented in Test 3 of Configuration 3 for the authentication server to read the key from the database server. To avoid being too redundant, we omit the process declaration. We then focus on comparing and analyzing the analysis results. For the SML code of the processes, please refer to the information in Appendix G.

Configuration 4:

(1) Responders are on the Internet; initiators, the authentication server and database server are located in the office network;

(2) Attackers' initial knowledge includes the name of all participants, the name of the office network and the name of the database server;

(3) Attackers can access to the Internet only.

We present the structure of the ambients as below figure according to the above assumptions.

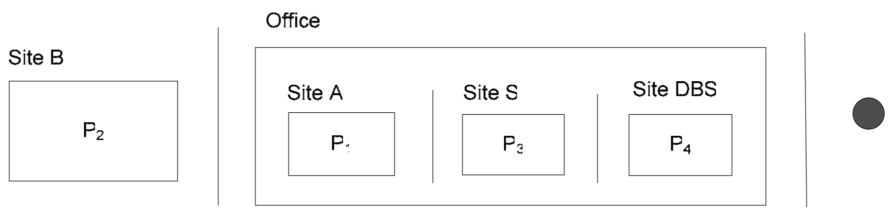


Figure 6-4: Ambient structure of Configuration 4

Test 1

We validate WMF protocol and its two variants under the above assumption. The analysis results are summarized in Table 6-6.

	WMF	WMF-V1	WMF-V2
$\varphi -$ component	$\emptyset$	$\{(A_{2i}, B_{2j}) \mid 1 \le i, j \le n\}$*	$\emptyset$*

Table 6-6: The experiment results for Test 1 of Configuration 4.

** The over-estimate has been removed in order to present a clear view about the status of authenticity of a protocol.*

The results for WMF and WMF-V1 are just as those of the case that both initiators and responders are on the Internet. The change happens at the WMF-V2. It is flawless now. Here one implicit assumption is necessary to clarify that malicious legal sites belong to the class of attackers since they are malicious. These bad behavior sites could be initiators or responders. This concept is implicit in LySa as there is only one network and thus attackers and participants of a protocol stay on the same location. In a hierarchical network, however, we must be careful which site could be malicious. Take the above descriptions of the configuration for example, it actually forces the condition that every initiator is friendly; otherwise, attackers should be supposed to be in the office network and our assumption is violated. When we analyze the result for the WMF-V2, this implicit condition is applied to determine which pair(s) of the $\varphi -$ component is(are) over-estimate.

Test 2

Observing that communication between *A* and *S* is transparent to attackers, we try to remove encryption of messages in the first step of WMF and extend the message with a unique number only known by *S* and *A*. The suggested protocol is described as:

$$\begin{aligned}
&1.\ A \to S:\quad u, A, B, K_{AB}\\
&2.\ S \to B:\quad K_{BS}[A, K_{AB}]\\
&3.\ A \to B:\quad K_{AB}[M]
\end{aligned}$$

There are three benefits given by the optimization: (1) This change saves the time for encrypting and decrypting the message in line 1; (2) It saves the space on the database server for storing the key K_{AS} ; (3) It saves the time for the authentication server to get the key K_{AS} from the database server.

Even though we can expect that the protocol is secure, the actual experiment on our computer unfortunately can not afford the heavy computation and reported the error – "the RAM is used up". This is because we have a four-tuple message (instead of three-tuple) that means the number of quantifications in ALFP formula is also increased. Suppose the number of the set of the universe $\mathcal{U}$ is *n*. Then increasing the number of quantification of a formula by one means the new computation time is up to n times as big as that of the former. Correspondingly, the space consumed by the solver is increased much too.

Configuration 5:

(1) Initiators are on the Internet; responders, the authentication server and database server are located in the office network;

(2) Attackers' initial knowledge includes the name of all participants, the name of the office network and the name of the database server;

(3) Attackers can access to the Internet only.

We present the structure of the ambients as below figure according to the above assumptions.

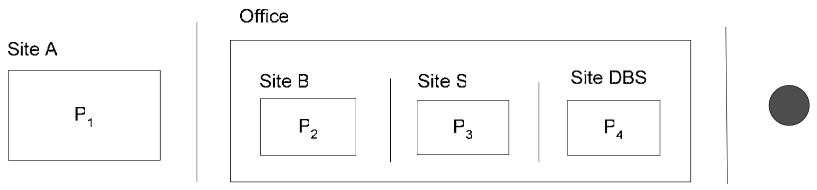


Figure 6-5: Ambient structure of Configuration 5

Test 1

The analysis results are summarized in Table 6-7.

	WMF	WMF-V1	WMF-V2
φ – component	$\emptyset$	$\emptyset$*	$\{(A_{2i}, B_{2j}) \mid 1 \le i, j \le n\}$ *

Table 6-7: **The experiment results for Test 1 of Configuration 5.**

** The over-estimate has been removed in order to present a clear view about the status of authenticity of a protocol.*

Compared with the results in Table 6-6, the new results reflect the exchange of the locations of initiators and responders. As expected, now the communication between the server and responders are protected from the eavesdropping of attackers. Similar to the idea in the Test 2 of Configuration 4, we can remove the message encryption and decryption in the step 2.

$$\begin{aligned}
&1.\ A \to S:\ \ A, K_{AS}[\,B, K_{AB}]\\
&2.\ S \to B:\ \ u, A, K_{AB}\\
&3.\ A \to B:\ \ K_{AB}[M]
\end{aligned}$$

The corresponding test shows that the optimized protocol is still secure with respect to authenticity.

Configuration 6:

(1) Initiators are on the Internet; responders, the authentication server and database server are located in the office network; responders are movable and they may work on the Internet;

(2) Attackers' initial knowledge includes the name of all participants, the name of the office network and the name of the database server;

(3) Attackers can access to the Internet only.

We present the structure of the ambients as below figure according to the above assumptions.

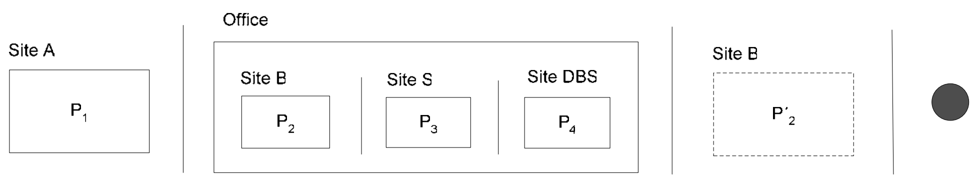


Figure 6-6: Ambient structure of Configuration 6

Test 1

As no sites can move in our calculus, we normally need declare two separate processes: one is for the case that B is inside office network; another is for the case that B is on the Internet. After analyzing the two processes and removing the over-estimates in their $\varphi-$ components individually, we unite the two $\varphi-$ components. This will be treated as the estimate for the whole configuration. However, one can expect that the result will be the same as the case that the responders are on the Internet only (described by Configuration 3). If the protocol is flawed in the case that the responder is in the local network, then it also flawed when the responder is on the Internet as the attacker can get more information now. Our experiment on WMF and its two variants also confirmed our conjecture: for the three protocols the estimates based on this configuration are same with those based on Configuration 3. The analysis results for the two configurations are summarized in Table 6-8.

	WMF	WMF-V1	WMF-V2
$\varphi-$ component	$\emptyset$	$\{(A_{2i}, B_{2j}) \mid 1 \le i, j \le n\}$*	$\{(A_{2i}, B_{2j}) \mid 1 \le i, j \le n\}$*

Table 6-8: The experiment results for Test 1 of Configuration 6.

** The over-estimate has been removed in order to present a clear view about the status of authenticity of a protocol.*

Configuration 7:

(1) Initiators are on the Internet; responders, the authentication server and database server are located in the office network;

(2) Attackers' initial knowledge includes the name of all participants, the name of the office network and the name of the database server;

(3) Attackers can access to the Internet and the office network.

The structure of the ambients is specified as below figure according to the above assumptions.

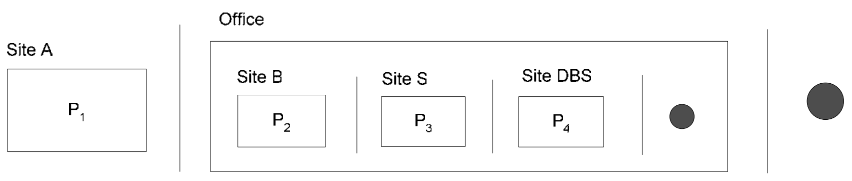


Figure 6-7: Ambient structure of Configuration 7

Test 1

This time attackers acquired full capability to eavesdrop on any packet transferred between any two participants. We can reasonably guess that this configuration will have same estimates with the configuration described in Figure 6-2. The guess is proved by our experiment again: For all of the WMF, WMF-V1 and WMF-V2, attackers know all keys including K_{AS} , K_{BS} and K_{AB} and thus the three protocols are completely cracked by the attacker.

Configuration 8:

(1) Responders are on the Internet; initiators are in the office network; the authentication server and database server are located in the computation center;

(2) Attackers' initial knowledge includes the name of all participants, the name of the office network, the name of the computation center and the name of the database server;

(3) Attackers can access to the Internet and the office network only.

The structure of the ambients is specified as below figure

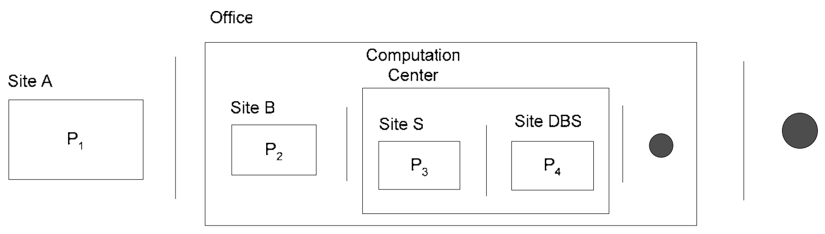


Figure 6-8: Ambient structure of Configuration 8

Test 1

The estimates for the three protocols are same as those in Table 6-8. As the two attackers share their knowledge each other, the effect of the boundary of the office network diminishes and thus in that sense site B can be thought to be on the Internet. This explains why we get the same result as the Test 1 of Configuration 6.

6.4 Summary

We have experimented on symmetric key protocols both in the flat space of networks and in hierarchical networks. Our estimate results show that our analysis works well for both of the two kinds of networks. In particular, our tool can safely approximate the behavior of the protocols which are applied on a hierarchical network. Some protocols are further optimized to improve their performance according to the specific network environment and assumptions. Our experience about the experiment can be summarized as below:

(1) As ambients can be used to formalize the objects which may be very different in properties considered in our model, such as local networks, data files, etc. one must be careful when locating the attacker to make sure the assumption is proper.

(2) The estimate computed by the analysis is an over-estimate. Therefore the skill of distinguishing between real answer and over-estimate is necessary. The two kinds of typical over-estimate in the analysis are illustrated in Test 1 of Configuration 3. As far as we know, they are the reason of all over-estimates found in our experiment.

7 Conclusion

In this chapter we summarize the key contributions and conclusions of this M.Sc. Thesis work that has been carried out in relation to the project “Static Analysis for Protocol Validation in Hierarchical Networks”. We end the chapter with the discussion of future work that could be performed to investigate interesting areas or lead to further clarifications or improvements on the existing work.

7.1 Overview of Contribution

The contribution of our work could be summarized as four aspects:

The calculus. We have investigated how encryptions and decryptions could be expressed in ABoxed Ambients. This is important for deciding where to add annotations declaring the origin and destination of encrypted messages in order to check authentication property. We have also presented how hierarchical networks and protocols applied on such networks can be formalized as ABoxed Ambients processes. Participants of protocols and local networks are modeled as ambients. Packets are also modeled as ambients and their ability to move around is governed by the mobility primitives possessed. Since ambient calculus is very expressive, it sometimes attempts to assign the modeled objects more capabilities than what they are supposed to have, and thus some properties of the object are changed or vanished. To describe the characteristic of objects more precisely, we remove the undesired capabilities from these objects. This was achieved by our well-formednesses which classify the ambients into the ambients of sites and the ambients of packets.

Static analysis. The static analysis has been developed for tracking the set of encrypted messages decrypted at each relevant point successfully. Since the notion of authentication is directly captured by the operational semantics, it can also be safely estimated by the analysis. We have also proved the correctness of the analysis with respect to the operational semantics in order to ensure that the analysis can safely estimate the behavior of a process throughout its execution.

Network attacker. Our attacker is based on the classic Dolev-Yao attacker. Moreover we have added necessary assumptions onto the Dolev-Yao condition to complaint with the environment of the hierarchical network. We then defined the adjusted Dolev-Yao condition. Last we have proved that the attacker is hardest with respect to the well-formednesses. Other kinds of attackers that may be modeled by ambient calculus have also been discussed.

The implementation and experiments on protocols. Following the implementation of the analysis, we have developed a tool in SML of New Jersey to automate our analysis. With the help of the tool, we validated many symmetric key protocols which are formalized as ABoxed Ambients processes. Our experiment results have shown that the analysis can pinpoint many kinds of errors that can be captured by LySa in the environment of hierarchical networks.

7.2 Future Work

Several areas or solutions have been examined in thought but have not been put into practice for the restriction on time. They are listed as below.

- **Extending the calculus with asymmetric keys.**

Our framework can be adapted to deal with perfect asymmetric cryptography as demonstrated in [2, chapter 8]. The idea can be sketched as follows. First we define the set of names for pairs of asymmetric keys m^+ and m^-:

$$\begin{array}{lll} C ::= n_s & \text{site name} & n_s \in \mathcal{C}_s \\ \quad \mid\ n & \text{non-site name} & n \in \mathcal{C}_P \\ \quad \mid\ m^+ & \text{non-site name} & m \in \mathcal{C}_P \\ \quad \mid\ m^- & \text{non-site name} & m \in \mathcal{C}_P \end{array}$$

The process *P* has two more constructs for creating a new pair of public/private keys.

$P ::=$

...

$$\begin{array}{ll} \mid\ (v\pm\ m)P & \text{key pair creation} \\ \mid\ (vk\pm\ m)P & \text{key pair creation which is transparent to the attacker} \end{array}$$

We shall say that $\lfloor m \rfloor, \lfloor m^+ \rfloor, \lfloor m^- \rfloor$ are pairwise distinct. Surely we need modify the rules for substitution and add more rules for congruence like LySa does. In particular, we add three additional reduction rules

(A-New) (A-News)

$$\frac{P \to_{\mathcal{R}} Q}{(v\pm\ m)P \to_{\mathcal{R}} (v\pm\ m)Q} \qquad \frac{P \to_{\mathcal{R}} Q}{(vk\pm\ m)P \to_{\mathcal{R}} (vk\pm\ m)Q}$$

$(A-$Output-Par2)

$$\frac{\wedge_{i=1}^{j} M_i = M_i' \quad \wedge \quad \mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})}{(M_1', \cdots, M_j'; x_{j+1}, \cdots, x_k)_{\ell'}^{m^+} [\text{orig } \mathcal{L}'].P \mid m^-[< M_1, \cdots, M_k >_{\ell}^{\circ} [\text{dest } \mathcal{L}] \mid R] \to_{\mathcal{R}} \\ P\{x_1 \leftarrow M_1\} \cdots \{x_k \leftarrow M_k\} \mid n[R]}$$

$(A-$Output-Par3)

$$\frac{\wedge_{i=1}^{j} M_i = M_i' \quad \wedge \quad \mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})}{(M_1', \cdots, M_j'; x_{j+1}, \cdots, x_k)_{\ell'}^{m^-} [\text{orig } \mathcal{L}'].P \mid m^+[< M_1, \cdots, M_k >_{\ell}^{\circ} [\text{dest } \mathcal{L}] \mid R] \to_{\mathcal{R}} \\ P\{x_1 \leftarrow M_1\} \cdots \{x_k \leftarrow M_k\} \mid n[R]}$$

Accordingly we can extend our analysis, attacker and implementation to subject to the changes in the calculus. Certainly the theorems or lemmas relating to the changes should be modified or extended too.

- **Optimizing the Flow Logic specification to improve the performance of the Succinct Solver.**

The actual formulation of the analysis has a great impact on the execution time on the solver. Two strategies are suggested in [34] to fasten the computation of the Succinct Solver. They are

(1) Adjusting the order of the parameters of relations.

The Succinct Solver uses prefix trees as an internal representation of relations [13]. Therefore, a generic strategy is to try to put the bound parameters ahead of the unbound ones. For example, assume that a relation $\gamma(x_1, x_2)$ is a precondition in which x_1 is always unbound but x_2 is always bound. Then querying the inverse relation $\gamma'(x_2, x_1)$ would be faster than querying $\gamma(x_1, x_2)$.

(2) Adjusting the order of conjunctions in preconditions.

Preconditions of an ALFP formula are evaluated from left to right. During the evaluation the environment which records the successful bindings of variables are kept updating. When checking a query to a relation γ, the evaluation of the remaining preconditions is performed under the updated environment that is calculated by unifying the current environment with the elements in γ. If the unification fails, then no further evaluation is needed. That is the earlier a unification fails, the less effort is in environment updating. Therefore we should arrange the query, which restricts the variable binding most, ahead of other preconditions. For example, for the clause

$$\forall x, \forall y : \gamma(x, y) \wedge Site(x) \Rightarrow \cdots$$

if *x* may be bound with more values in relation γ than those in relation *Site*, then reordering the preconditions as

$$\forall x, \forall y : Site(x) \wedge \gamma(x, y) \Rightarrow \cdots$$

is more efficient.

As we have one incomplete experiment whose computation is too large to be afforded by our computer, we expect the optimizations of the specification can reduce the size of computation and complete the experiment.

- **Defining other attackers**

Our attacker is basic among the attackers could be modeled by the ambient calculus. Although we discussed three more powerful attackers in Chapter 4, what security property these attackers may be useful to validate need further investigation. Still more other attackers not listed in this thesis may exist and are waiting for us to explore.

Appendix A Protocol Narrations

Wide Mouthed Frog. [31]

1. $A \to S: \; A, K_{AS}[B, K_{AB}]$
2. $S \to B: \; K_{BS}[A, K_{AB}]$
3. $A \to B: \; K_{AB}[M]$

The initiator's name is not encrypted:
2. $S \to B: \; A, K_{BS}[K_{AB}]$
The responder's name is not encrypted:
1. $A \to S: \; A, B, K_{AS}[K_{AB}]$

NeedHam-Schroeder (symmetric key). [30]

1. $A \to S: \; A,\; B,\; R_A$
2. $S \to A: \; K_{AS}[R_A, B, K_{AB}, K_{BS}[A, K_{AB}]]$
3. $A \to B: \; K_{BS}[A, K_{AB}]$
4. $B \to A: \; K_{AB}[R_B]$
5. $A \to B: \; K_{AB}[R_B - 1]$
6. $A \to B: \; K_{AB}[M]$

Correcting the flaw
4. $B \to A: \; K_{AB}[u_1, R_B]$
5. $A \to B: \; K_{AB}[u_2, R_B - 1]$

Otway-Rees. [27]

1. $A \to B: \; N, K_{AS}[N_A, N, A,\; B]$
2. $B \to S: \; N, K_{AS}[N_A, N, A,\; B], K_{BS}[N_B, N, A,\; B]$
3. $S \to B: \; N, K_{AS}[N_A, K_{AB}], K_{BS}[N_B, K_{AB}]$
4. $B \to A: \; N, K_{AS}[N_A, K_{AB}]$
5. $A \to B: \; K_{AB}[M]$

Yahalom. [29]

1. $A \to B: \; A, N_A$
2. $B \to S: \; B, K_{BS}[A, N_A, N_B]$
3. $S \to A: \; K_{AS}[B, K_{AB}, N_A, N_B], K_{BS}[A, K_{AB}]$
4. $A \to B: \; K_{BS}[A, K_{AB}], K_{AB}[N_B]$
5. $A \to B: \; K_{AB}[M]$

Andrew Secure RPC. [28]

1. $A \rightarrow B$: $A, K_{AB}[N_A]$
2. $B \rightarrow A$: $K_{AB}[N_A + 1, N_B]$
3. $A \rightarrow B$: $K_{AB}[N_B + 1]$
4. $B \rightarrow A$: $K_{AB}[K'_{AB}, N'_B]$
5. $A \rightarrow B$: $K'_{AB}[M]$

Correcting the flaw:
1. $A \rightarrow B$: $A, K_{AB}[u_1, N_A]$
2. $B \rightarrow A$: $K_{AB}[u_2, N_A + 1, N_B]$
3. $A \rightarrow B$: $K_{AB}[u_3, N_B + 1]$
4. $B \rightarrow A$: $K_{AB}[u_4, K'_{AB}, N'_B]$

Appendix B Validation of Other Protocols

B.1 Validating Otway-Rees.

The extended protocol narration of Otway-Rees is specified as

1. $A \rightarrow \ : N, K_{AS}[N_A, N, A, B]$[dest S] $\quad$ {assuming N, N_A are new values}

   $\rightarrow B : \ z_N, z_1$

2. $B \rightarrow \ : z_N, z_1, K_{BS}[N_B, z_N, A, B]$[dest S] $\quad$ {assuming N_B is new value}

   $\rightarrow S : y_N, y_1, y_2 K_{AS}[N_A, N, A, B], K_{BS}[N_B, N, A, B]$

   S : decrypt y_1 as $K_{AS}[y_{N_A}, y'_N, y_A, y_B]$[orig A] $\quad$ {check $y_N = y'_N, y_A = A$ and $y_B = B$}

   decrypt y_2 as $K_{BS}[y_{N_B}, y''_N, y'_A, y'_B]$[orig B] $\quad$ {check $y''_N = y'_N, y'_A = A$ and $y'_B = B$}

3. $S \rightarrow \ : y_N, K_{AS}[y_{N_A}, K_{AB}]$[dest A]$, K_{BS}[y_{N_B}, K_{AB}]$[dest B] $\quad$ {assuming K_{AB} is new value}

   $\rightarrow B : z'_N, z_2, z_3$ $\quad$ {check $z_N = z'_N$ }

   B : decrypt z_3 as $K_{BS}[z_{N_B}, z_{K_{AB}}]$[orig S] $\quad$ {check $z_{N_B} = N_B$}

4. $B \rightarrow \ : z_N, z_2$

   $\rightarrow A : x_N, x_1$ $\quad$ {check $x_N = N$}

   B : decrypt x_1 as $K_{AS}[x_{N_A}, x_{K_{AB}}]$[orig S] $\quad$ {check $x_{N_A} = N_A$}

5. $A \rightarrow \ : \ x_{K_{AB}}[M]$[dest B] $\quad$ {assuming is M a new value}

   $\rightarrow B : \ z_4$

   B : decrypt z_4 as $z_{K_{AB}}[z_M]$[orig A]

The ABoxed Ambients specification of the protocol is accordingly declared as

$(v\ K_{AS})(v\ K_{BS})$

$(A[(v\ N)\ (v\ N_A)\ (K_{AS}[\text{out } A.\text{ in } B.\ (<N>^{\uparrow} |\ \text{out B.in S.}(<N>^{\uparrow} | <A, B, N, N_A>^{\circ}_{A_1}\ [\text{dest S}_1]))]\ |$

$\qquad (N;)^{\circ}.(N_A; x_{K_{AB}})^{K_{AS}}_{A_2}[\text{orig S}_4].$

$\qquad (v\ M)x_{K_{AB}}[\text{out } A.\text{ in } B.\ <M>^{\circ}_{A_3}[\text{dest B}_3]]\ )]$

$|$

$S[(;y_N)^{\circ}.\ (A, B, y_N; y_{N_A})^{K_{AS}}_{S_1}[\text{orig A}_1].(A, B, y_N; y_{N_B})^{K_{BS}}_{S_2}[\text{orig B}_1].(v\ K_{AB})$

$\qquad K_{BS}[\text{out } S.\text{ in } B.\ (<y_N>^{\uparrow} | <y_{N_B}, K_{AB}>^{\circ}_{S_3}\ [\text{dest B}_2]\ |$

$\qquad\qquad K_{AS}[\text{out } K_{BS}.\text{out } B.\text{in } A.(<y_N>^{\uparrow} | <y_{N_A}.K_{AB}>^{\circ}_{S_4}\ [\text{dest A}_2]])]]]$

$|$

$B[(;z_N)^{\circ}.(v\ N_B)(K_{BS}[\text{out } B.\text{ in } S. <A, B, z_N, N_B>^{\circ}_{B_1}\ [\text{dest S}_2]]\ |$

$\qquad (;z_N)^{\circ}.(N_B; z_{K_{AB}})^{K_{BS}}_{B_2}[\text{orig S}_3].(;z_M)^{z_{K_{AB}}}_{B_3}[\text{orig A}_3].\cdots z_M \cdots)])$

$|\ \bullet$

The experiment result shows there is no authentication violation in Otwee-Rees as expected. But a phenomenon caused our attention: The computing time of the analysis for the protocol is apparently longer than the one of WMF and Needham-Schroeder. By analyzing the result of Otwee-Rees we found a large number of four-tuple messages and it is believed that the most of them are created by $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ attackers. Thus we conclude that the longer the arity is, the more messages attackers can make.

B.2 Validating Yahalom.

The extended protocol narration of Yahalom is specified as

1. $A \rightarrow$: A, N_A {assuming N_A is a new value}
   $\rightarrow B$: z_A, z_{N_A} {assuming $z_A = A$}
2. $B \rightarrow$: $B, K_{BS}[z_A, z_{N_A}, N_B]$[dest S] {assuming N_B is a new value}
   $\rightarrow S$: y_B, y_1 {check $y_B = B$}
   S : decrypt y_1 as $K_{BS}[y_A, y_{N_A}, y_{N_B}]$[orig B] {check $y_A = A$}
3. $S \rightarrow$: $K_{AS}[y_B, K_{AB}, y_{N_A}, y_{N_B}]$[dest A], $K_{BS}[y_A, K_{AB}]$[dest B] {assuming K_{AB} is a new value}
   $\rightarrow A$: x_1, x_2
   A : decrypt x_1 as $K_{AS}[x_B, x_{K_{AB}}, x_{N_A}, x_{N_B}]$[orig S] {check $x_B = B$}
4. $A \rightarrow$: $x_2, x_{K_{AB}}[x_{N_B}]$[dest B]
   $\rightarrow B$: z_1, z_2
   B : decrypt z_1 as $K_{BS}[z_A, z_{K_{AB}}]$[orig S] {check $z_A = A$}
   decrypt z_2 as $K_{AB}[z_{N_B}]$[orig A]
5. $A \rightarrow$: $x_{K_{AB}}[M]$[dest B] {assuming M is a new value}
   $\rightarrow B$: z_3
   B : decrypt z_3 as $z_{K_{AB}}[z_M]$[orig A]

Based on the above extended protocol narration, we program the protocol below.

$(v\ K_{AS})(v\ K_{BS})$

$(A[(v\ N_A)\ p[\text{out } A.\text{ in } B.\ < A,\ N_A >^{\uparrow}]\ |$

$\quad (B, N_A; x_{K_{AB}}, x_{N_B})_{A_1}^{K_{AS}} [\text{orig S}_2].x_{K_{AB}}[\text{out } A.\text{ in } B. < x_{N_B} >_{A_2}^{\circ} [\text{dest B}_3]]$

$\quad (x_{R_B})_{A_2}^{x_{K_{AB}}} [\text{orig B}_2].x_{K_{AB}}[\text{out A. in B. } < x_{N_B} >_{A_3}^{\circ} [\text{dest B}_3]]\ |$

$\quad (v\ M)x_{K_{AB}}[\text{out } A.\text{ in } B. <M>_{A_4}^{\circ} [\text{dest B}_4]]\ )]$

$|$

$S[(B;)^{\circ}.(A; y_{N_A}, y_{N_B})_{S_1}^{K_{BS}} [\text{dest B}_1](v\ K_{AB})$

$\quad K_{AS}[\text{out } S.\text{ in } A.\ (< B,\ y_{N_A}, K_{AB}, y_{N_B} >_{S_2}^{\circ} [\text{dest A}_2]\ |$

$\qquad K_{BS}[\text{out } K_{AS}.\text{out } A.\text{in } B. < A, K_{AB} >_{S_3}^{\circ} [\text{dest B}_2]])]]$

$|$

$B[(A; z_{N_A})^{\circ}.(v\ N_B)\ K_{BS}[\text{out } B.\text{ in } S.(< B >^{\uparrow} |< A, z_{N_A}, N_B >_{B_1}^{\circ} [\text{dest S}_1])]\ |$

$\quad (A; z_{K_{AB}})_{B_2}^{K_{BS}} [\text{orig S}_3].\underline{(N_B;)_{B_3}^{z_{K_{AB}}}} [\text{orig A}_3].\underline{(;z_M)_{B_4}^{z_{K_{AB}}}} [\text{orig A}_4].\cdots z_M \cdots])$

$|\ \bullet$

The $\varphi-$component of the estimate reports a pair which is actually an over-estimation.

B.3 Validating Andrew Secure RPC.

We expand the protocol narration for Andrew Secure RPC as

1. $A \rightarrow\ : A, K_{AB}[N_A][\text{dest B}]$ $\{\text{assuming } N_A \text{ is a new value}\}$
   $\rightarrow B\ :\ y_A, y_1$ $\{\text{check } y_A = A\}$
   $B\ :\ \text{decrypt } y_1 \text{ as } K_{AB}[y_{N_A}][\text{orig A}]$
2. $B \rightarrow\ : K_{AB}[y_{N_A} + 1, N_B][\text{dest A}]$ $\{\text{assuming } N_B \text{ is a new value}\}$
   $\rightarrow A\ :\ x_1$
   $A\ :\ \text{decrypt } x_1 \text{ as } K_{AB}[x_{N_A+1}, x_{N_B}][\text{orig B}]$ $\{\text{check } x_{N_A+1} = N_A + 1\}$
3. $A \rightarrow\ : K_{AB}[x_{N_B} + 1][\text{dest B}]$
   $\rightarrow B\ :\ y_2$
   $B\ :\ \text{decrypt } y_2 \text{ as } K_{AB}[y_{N_B+1}][\text{orig A}]$ $\{\text{check } y_{N_B+1} = N_B + 1\}$
4. $B \rightarrow\ :\ K_{AB}[K'_{AB}, N'_B][\text{dest A}]$ $\{\text{assuming } K'_{AB} \text{ and } N'_B \text{ are new values}\}$
   $\rightarrow A:\ x_2$
   $A\ :\ \text{decrypt } x_2 \text{ as } K_{AB}[x_{K'_{AB}}, x_{N'_B}]$
5. $A \rightarrow\ :\ x_{K'_{AB}}[M][\text{dest B}]$ $\{\text{assuming is } M \text{ a new value}\}$
   $\rightarrow B\ :\ y_3$
   $B\ : \text{decrypt } y_3 \text{ as } z_{K_{AB}}[z_M][\text{orig A}]$

The corresponding specification in ABoxed Ambients is coded as

$(v\ K_{AB})$

$(A[(v\ N_A)\ K_{AB}[\text{out } A.\text{ in } B.\ (<A>^{\uparrow} | <N_A>^{\circ}_{A_1}\ [\text{dest B}_1])] \ |$

$\quad (N_A, N_A; x_{N_B})^{K_{AB}}_{A_2}[\text{orig B}_2].K_{AB}[\text{out } A.\text{ in } B.\ <x_{N_B}, x_{N_B}>^{\circ}_{A_3}\ [\text{dest B}_3])] \ |$

$\quad (;x_{K'_{AB}}, x_{N'_B})^{K_{AB}}_{A_4}[\text{orig B}_4].\ (v\ M)x_{K'_{AB}}[\text{out } A.\text{ in } B.\ <M>^{\circ}_{A_5}[\text{dest B}_5]]\ )]$

$|$

$B[(A;)^{\circ}.(;y_{N_A})^{K_{AB}}_{B_1}[\text{orig A}_1].(v\ N_B)\ (K_{AB}[\text{out } B.\text{ in } A. <y_{N_A}, y_{N_A}, N_B>^{\circ}_{B_2}\ [\text{dest A}_2]\ ]|$

$\quad (N_B, \mathrm{N}_B;)^{K_{AB}}_{B_3}[\text{orig A}_3].(v\ K'_{AB})(v\ N'_B)\ (K_{AB}[\text{out } B.\text{ in } A. <K'_{AB}, N'_B>^{\circ}_{B_4}\ [\text{dest A}_4]\ ]|$

$\quad (;y_M)^{K'_{AB}}_{B_5}[\text{orig A}_5].\cdots y_M \cdots))])$

$| \ \bullet$

Similar to Needham-schroeder, Andrew Secure RPC has successor operation. And the operation is encoded by the pair of a number. The estimate is summarized as

$$\{(A_{3i}, A_{4i}) \mid 1 \le i \le n\}$$

Correcting the flaw The problem is quite similar to the flaw reported for Needham-schroeder. The way for fixing the flaw, therefore, is also similar: Add extra components $(u_1, u_2, \cdots)$ in the encrypted messages. The extended narration for the corrected steps is presented below.

1. $A \rightarrow\ : A, K_{AB}[u_1, N_A][\text{dest B}]$ $\quad \{\text{assuming } N_A \text{ is a new value}\}$
   $\rightarrow B\ : y_A, y_1$ $\quad \{\text{check } y_A = A\}$
   $B\ :\ \text{decrypt y}_1 \text{ as } K_{AB}[y_{u_1}, y_{N_A}][\text{orig A}]$ $\quad \{\text{check } y_{u_1} = u_1\}$
2. $B \rightarrow\ : K_{AB}[u_2, y_{N_A} + 1, N_B][\text{dest A}]$ $\quad \{\text{assuming } N_B \text{ is a new value}\}$
   $\rightarrow A\ :\ x_1$
   $A\ :\ \text{decrypt } x_1 \text{ as } K_{AB}[x_{u_2}, x_{N_A+1}, x_{N_B}][\text{orig B}]$ $\quad \{\text{check } x_{u_2} = u_2 \text{ and } x_{N_A+1} = N_A + 1\}$
3. $A \rightarrow\ : K_{AB}[u_3, x_{N_B} + 1][\text{dest B}]$
   $\rightarrow B\ : y_2$
   $B\ :\ \text{decrypt y}_2 \text{ as } K_{AB}[y_{u_3}, y_{N_B+1}][\text{orig A}]$ $\quad \{\text{check } y_{u_3} = u_3 \text{ and } y_{N_B+1} = N_B + 1\}$
4. $B \rightarrow\ :\ K_{AB}[u_4, K'_{AB}, N'_B][\text{dest A}]$ $\quad \{\text{assuming } K'_{AB} \text{ and } N'_B \text{ are new values}\}$
   $\rightarrow A:\ x_2$
   $A\ :\ \text{decrypt } x_2 \text{ as } K_{AB}[x_{u_4}, x_{K'_{AB}}, x_{N'_B}]$ $\quad \{\text{check } x_{u_4} = u_4\}$

The experiment on the modified protocol shows the static authentication success. This should also guarantee the dynamic authentication.

Appendix C Generation Function of the Attacker

The generation function $\mathcal{G}(\bullet, *)$ is written in ALFP logic as the conjunction $\wedge$ of below formulae.

1.1 $\wedge_{k\in A_k^+} \forall v_1,...,v_k : \kappa(*,v_1,...,v_k,\varepsilon,AW) \Rightarrow \ \wedge_{i=1}^{k} \text{genCap}(v_i)$

1.2 $\wedge_{k\in A_k^+} \forall \mu,v_1,...,v_k,\ell,l_c : \gamma(*,\mu) \wedge \text{Pack}(\mu) \wedge \kappa(\mu,v_1,...,v_k,\ell,l_c)$

$\Rightarrow \rho(t_\bullet,\mu,v_1,...,v_k,\ell,l_c)$

2 $\wedge_{k\in A_k^+} \forall \mu,v_1,...,v_k,\ell,l_c : \rho(t_\bullet,\mu,v_1,...,v_k,\ell,l_c) \wedge \rho(z_\bullet,\mu) \Rightarrow$

$\wedge_{i=1}^{k} \rho(z_\bullet,v_i) \wedge ((l_c \neq \text{AW}) \Rightarrow \varphi(\ell_\bullet,\ell))$

3.1 $\wedge_{k\in A_k^+} \forall v_1,...,v_k,\mu : \wedge_{i=1}^{k} \rho(z_\bullet,v_i) \wedge \rho(z_\bullet,\mu) \wedge \mu \neq in(m) \ \wedge \ \mu \neq out(m) \wedge \text{Pack}(\mu)$

$\Rightarrow \ \rho(t_\bullet,\mu,v_1,...,v_k,\ell_\bullet,\text{AW})$

3.2 $\wedge_{k\in A_k^+} \forall v_1,...,v_k,\mu : \wedge_{i=1}^{k} \rho(z_\bullet,v_i) \wedge \text{Reach}(*,\mu)$

$\Rightarrow \ \rho(t_\bullet,\mu,v_1,...,v_k,\varepsilon,\text{AW})$

4.1.1 $\forall t,m : \rho(z_\bullet,t) \ \wedge \ t \neq \text{in}(m) \ \wedge \ t \neq \text{out}(m) \wedge \text{Pack}(t) \wedge \forall \mu\text{: Reach}(*,\mu)$

$\Rightarrow \gamma(\mu,t)$

4.2 $\wedge_{k\in A_k^+} \forall \mu,v_1,...,v_k,\ell,l_c : \rho(t_\bullet,\mu,v_1,...,v_k,\ell,l_c) \Rightarrow \ \kappa(\mu,v_1,...,v_k,\ell,l_c)$

5.1 $\rho(z_\bullet,n_\bullet) \wedge \rho(z_\bullet,n_\bigstar) \wedge (\forall t : \mathcal{N}_{\text{f}}(t) \Rightarrow \ \rho(z_\bullet,t)) \wedge (\forall t : \mathcal{N}_{\text{v}}^{*}(t) \Rightarrow \ \rho(z_\bullet,t))$

5.2 $\rho(z_\bullet,\text{in}(n_\bullet)) \wedge \rho(z_\bullet,\text{out}(n_\bullet))$

$(\forall t : \mathcal{N}_{\text{f}}(t) \Rightarrow \rho(z_\bullet,\text{in}(t)) \wedge \rho(z_\bullet,\text{out}(t))) \wedge$

$(\forall t : \mathcal{N}_{\text{v}}^{*}(t) \Rightarrow \rho(z_\bullet,\text{in}(t)) \wedge \rho(z_\bullet,\text{out}(t)))$

The auxiliary function Reach and genCap are transformed as the conjunction of the below formulae.

Formulae for **Reach**	Formulae for **genCap**
1 $\forall \mu : \text{Site}(\mu) \Rightarrow \text{Reach}(\mu,\mu)$ 2 $\forall \mu_1,\mu_2 : \text{Site}(\mu_1) \wedge \text{Site}(\mu_2) \wedge$ $\rho(z_\bullet,\mu_1) \wedge \rho(z_\bullet,\mu_2) \wedge \gamma(\mu_1,\mu_2)$ $\Rightarrow \text{Reach}(\mu_1,\mu_2)$ 3 $\forall \mu_1,\mu_2 : \text{Site}(\mu_1) \wedge \text{Site}(\mu_2) \wedge$ $\text{Reach}(\mu_1,\mu_2)$ $\Rightarrow \text{Reach}(\mu_2,\mu_1)$ 4 $\forall \mu_1, \mu_2, \mu_3\text{: Reach}(\mu_1,\mu_2) \wedge \text{Reach}(\mu_2,\mu_3)$ $\Rightarrow \text{Reach}(\mu_1,\mu_3)$	1 $\forall t : v_i = \text{in}(t)$ $\Rightarrow \rho(z_\bullet,\text{out}(t)) \wedge \rho(z_\bullet,t)$ 2 $\forall t : v_i = \text{out}(t)$ $\Rightarrow \rho(z_\bullet,\text{in}(t)) \wedge \rho(z_\bullet,t)$ 3 $\forall t : v_i \neq \text{in}(t) \wedge v_i \neq \text{out}(t)$ $\Rightarrow \rho(z_\bullet,\text{out}(v_i)) \wedge \rho(z_\bullet,\text{in}(v_i))$

Appendix D Proofs

D.1 Correctness of the Analysis

Theorem 1: Subject reduction
If $P \rightarrow_{\mathcal{R}} Q$ and $(\gamma,\kappa,\rho) \models^{*} P : \varphi$ then $(\gamma,\kappa,\rho) \models^{*} Q : \varphi$.

Proof. We prove the more general result

- If $P \rightarrow_{\mathcal{R}} Q$ and $(\gamma,\kappa,\rho) \models^{*} P : \varphi$ then $(\gamma,\kappa,\rho) \models^{*} Q : \varphi$; furthermore, if $\varphi = \emptyset$ then $P \rightarrow_{\mathrm{RM}} Q$.

The proof proceeds by structural induction in the reduction steps.
Case (New). Assume that $(\gamma,\kappa,\rho) \models^{*} (\nu\ n)P : \varphi$ i.e. that $(\gamma,\kappa,\rho) \models^{*} P : \varphi$. Assume also that $(\nu\ n)P \rightarrow (\nu\ n)P'$ using (New) because $P \rightarrow P'$. Then by the induction hypothesis $(\gamma,\kappa,\rho) \models^{*} P' : \varphi$, which by the analysis definition allows to conclude $(\gamma,\kappa,\rho) \models^{*} (\nu\ n)P' : \varphi$.

Case (News) is similar to the case (New).

Case (Amb). Assume that $(\gamma,\kappa,\rho) \models^{*} n[P] : \varphi$ i.e. that $\forall \mu \in \mathcal{N}_{\rho}(n) : \mu \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\mu} P : \varphi$. As $\mathcal{N}_{\rho}(n)$ is $\{\lfloor n \rfloor\}$, we can write the formula equivalently to be $\lfloor n \rfloor \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} P : \varphi$ (where n is its canonical version). We further assume that $n[P] \rightarrow n[P']$ by (Amb) because $P \rightarrow P'$. Then by the induction hypothesis $(\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} P' : \varphi$, which together with $\lfloor n \rfloor \in \gamma(*)$ and $\mathcal{N}_{\rho}(n) = \lfloor n \rfloor$ allows to conclude $(\gamma,\kappa,\rho) \models^{*} n[P'] : \varphi$ by the analysis definition.

Case (Par). Assume that $(\gamma,\kappa,\rho) \models^{*} P_1 \mid P_2 : \varphi$ i.e. that $(\gamma,\kappa,\rho) \models^{*} P_1 : \varphi$ and $(\gamma,\kappa,\rho) \models^{*} P_2 : \varphi$. Furthermore, assume that $P_1 \mid P_2 \rightarrow P_1' \mid P_2$ by (Par) because $P \rightarrow P'$. The applying the induction hypothesis $(\gamma,\kappa,\rho) \models^{*} P' : \varphi$. The analysis allows to conclude that $(\gamma,\kappa,\rho) \models^{*} P_1' \mid P_2 : \varphi$.

Case (Congr) is directly given by the induction hypothesis and Lemma 2.

Case (In). Let $P = m[\mathrm{in}\ n.P_1 \mid Q] \mid n[P_2]$ and $P' = n[m[P_1 \mid Q] \mid P_2]$ where $m \in \mathcal{C}_p$ and assume that $(\gamma,\kappa,\rho) \models^{*} P : \varphi$ and that $P \rightarrow P'$ due to (In). We have to prove $(\gamma,\kappa,\rho) \models^{*} P' : \varphi$. Expanding the analysis, we get

$$
\begin{aligned}
&(\gamma,\kappa,\rho) \models^* P : \varphi \\
&\text{iff} \quad (\gamma,\kappa,\rho) \models^* m[\text{in } n.P_1 \mid Q] : \varphi \wedge (\gamma,\kappa,\rho) \models^* n[P_2] : \varphi \\
&\text{iff} \quad \forall \mu \in \mathcal{N}_\rho(m) : \mu \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^\mu \text{in } n.P_1 \mid Q : \varphi \wedge \\
&\qquad \wedge (\gamma,\kappa,\rho) \models^* n[P_2] : \varphi \\
&\text{iff} \quad \forall \mu \in \mathcal{N}_\rho(m) : \mu \in \gamma(*) \wedge \mathcal{M}_\rho(\text{in } n) \subseteq \gamma(\mu) \wedge (\gamma,\kappa,\rho) \models^\mu P_1 \mid Q : \varphi \wedge \\
&\qquad \wedge \forall \text{in } \mu' \in \mathcal{M}_\rho(\text{in } n) : \varphi_{\text{in}}(\mu') \wedge (\gamma,\kappa,\rho) \models^* n[P_2] : \varphi \\
&\text{iff} \quad \forall \mu \in \mathcal{N}_\rho(m) : \mu \in \gamma(*) \wedge \mathcal{M}_\rho(\text{in } n) \subseteq \gamma(\mu) \wedge (\gamma,\kappa,\rho) \models^\mu P_1 \mid Q : \varphi \wedge \\
&\qquad \wedge \forall \text{in } \mu' \in \mathcal{M}_\rho(\text{in } n) : \varphi_{\text{in}}(\mu') \wedge \\
&\qquad \forall \mu'' \in \mathcal{N}_\rho(n) : \mu'' \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\mu''} P_2 : \varphi
\end{aligned}
$$

The φ_{in} closure condition gives at least one more constraint $\lfloor m \rfloor \in \gamma(\lfloor n \rfloor)$. Since by $\mathcal{N}_\rho(n) = \{\lfloor n \rfloor\}$, $\mathcal{N}_\rho(m) = \{\lfloor m \rfloor\}$ and $\mathcal{M}_\rho(\text{in } n) = \{\text{in } \lfloor n \rfloor\}$, we equivalently rewritten the above analysis as

$$
\begin{aligned}
&\lfloor m \rfloor \in \gamma(*) \wedge (\text{in } \lfloor n \rfloor) \subseteq \gamma(\lfloor m \rfloor) \wedge \varphi_{\text{in}}(\lfloor m \rfloor) \wedge \\
&\lfloor m \rfloor \in \gamma(\lfloor n \rfloor) \wedge (\gamma,\kappa,\rho) \models^{\lfloor m \rfloor} P_1 \mid Q : \varphi \wedge \\
&\lfloor n \rfloor \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} P_2 : \varphi
\end{aligned}
\qquad (1)
$$

By the analysis of the ambient and the fact $\mathcal{N}_\rho(m) = \{\lfloor m \rfloor\}$, we have

$$
\begin{aligned}
&\lfloor m \rfloor \in \gamma(\lfloor n \rfloor) \wedge (\gamma,\kappa,\rho) \models^{\lfloor m \rfloor} P_1 \mid Q : \varphi \\
&\text{iff} \quad \forall \mu \in \mathcal{N}_\rho(m) : \mu \in \gamma(\lfloor n \rfloor) \wedge (\gamma,\kappa,\rho) \models^\mu P_1 \mid Q : \varphi \\
&\text{iff} \quad (\gamma,\kappa,\rho) \models^n m[P_1 \mid Q] : \varphi
\end{aligned}
\qquad (2)
$$

The analysis of the parallelism then gives

$$
\begin{aligned}
&(\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} m[P_1 \mid Q] : \varphi \wedge (\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} P_2 : \varphi \\
&\text{iff} \quad (\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} m[P_1 \mid Q] \| P_2 : \varphi
\end{aligned}
\qquad (3)
$$

By the analysis of the ambient and the fact $\mathcal{N}_\rho(n) = \{\lfloor n \rfloor\}$, we have

$$
\begin{aligned}
&\lfloor n \rfloor \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\lfloor n \rfloor} m[P_1 \mid Q] \| P_2 : \varphi \\
&\text{iff} \quad \forall \mu \in \mathcal{N}_\rho(n) : \mu \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^\mu m[P_1 \mid Q] \| P_2 : \varphi \\
&\text{iff} \quad (\gamma,\kappa,\rho) \models^* n[m[P_1 \mid Q] \| P_2] : \varphi
\end{aligned}
\qquad (4)
$$

Applying (2), (3) and (4) on (1) sequentially, we finally get

$$
\begin{aligned}
&\lfloor m \rfloor \in \gamma(*) \wedge (\text{in } \lfloor n \rfloor) \subseteq \gamma(\lfloor m \rfloor) \wedge \varphi_{\text{in}}(\lfloor m \rfloor) \wedge \\
&(\gamma,\kappa,\rho) \models^* n[m[P_1 \mid Q] \| P_2] : \varphi
\end{aligned}
$$

It then gives $(\gamma,\kappa,\rho) \models^* n[m[P_1 \mid Q] \| P_2] : \varphi$ i.e. that $(\gamma,\kappa,\rho) \models^* P' : \varphi$ as required.

Case (Out) is similar to the case (In).

Case (LocCom). Let $P = <\mathrm{M}_1,\cdots,\mathrm{M}_\mathrm{k}>^{\circ}_{\ell}[\mathrm{dest}\ \mathcal{L}] \mid (M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\circ}.P_1$ and $P' = P_1\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\}$ and assume that $(\gamma,\kappa,\rho) \vDash^* P:\varphi$ and that $P \to P'$ because of (LocCom). We have to prove $(\gamma,\kappa,\rho) \vDash^* P':\varphi$. Unfolding the analysis one gets

$(\gamma,\kappa,\rho) \vDash^* P:\varphi$

iff $(\gamma,\kappa,\rho) \vDash^* <M_1,\cdots,M_k>^{\circ}_{\ell}[\mathrm{dest}\ \mathcal{L}]:\varphi \wedge (\gamma,\kappa,\rho) \vDash^* (M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\circ}.P:\varphi$

iff $\forall v_1,\cdots,v_k\colon \wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i) \Rightarrow <v_1,\cdots,v_k>_{\ell}[\mathrm{dest}\ \mathcal{L}] \in \kappa(*) \wedge$ (1)

$<v_1,\cdots,v_k>_{\ell}[\mathrm{dest}\ \mathcal{L}] \in \kappa(*)\colon \wedge_{i=1}^{j} v_i \in \mathcal{M}_\rho(M'_i)$ (2)

$\Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(\lfloor x_i \rfloor) \wedge$ (3)

$(\neg\mathrm{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L}) \Rightarrow (\ell,\varepsilon) \in \varphi) \wedge (\gamma,\kappa,\rho) \vDash^* P_1{:}\varphi$ (4)

Since fv(M_i) = $\emptyset$ then (1) gives $<\lfloor M_1 \rfloor,\cdots,\lfloor M_k \rfloor>_{\ell}[\mathrm{dest}\ \mathcal{L}] \in \kappa(*)$. The assumption $\wedge_{i=1}^{j} M_i = M'_i$ then gives $\wedge_{i=1}^{j} \lfloor M_i \rfloor \in \mathcal{M}_\rho(M'_i)$. Now from (3) and (4) we get $\wedge_{i=j+1}^{k} \lfloor M_i \rfloor \in \rho(\lfloor x_i \rfloor)$ and $(\gamma,\kappa,\rho) \vDash^* P_1{:}\varphi$. The Lemma 1 then gives $(\gamma,\kappa,\rho) \vDash^* P_1\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\}{:}\varphi$ i.e. that $(\gamma,\kappa,\rho) \vDash^* P':\varphi$ as required. For the second part of the result we see that $\neg\mathrm{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L}) \Rightarrow (\ell,\varepsilon) \in \varphi$ follows from (2) and since $\varphi = \emptyset$ it must be the case that $\mathrm{RM}(\ell,\mathcal{D},\varepsilon,\mathcal{L})$. Therefore the conditions of the rule (LocCom) are met for $\to_{\mathrm{RM}}$.

Case (Output-Chd1). Let $P = <\mathrm{M}_1,\cdots,\mathrm{M}_\mathrm{k}>^{n} \mid n[(M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\circ}.P_1 \mid \mathrm{Q}]$ and $P' = n[P_1\{x_1 \leftarrow M_1\}\cdots\{x_k \leftarrow M_k\} \mid \mathrm{Q}]$ and assume that $(\gamma,\kappa,\rho) \vDash^* P:\varphi$ and that $P \to P'$ because of (Output-Chd1). We have to prove $(\gamma,\kappa,\rho) \vDash^* P':\varphi$. Expanding the analysis one gets

$(\gamma,\kappa,\rho) \vDash^* P:\varphi$

iff $(\gamma,\kappa,\rho) \vDash^* <M_1,\cdots,M_k>^{n}:\varphi \wedge (\gamma,\kappa,\rho) \vDash^* n[(M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\circ}.P_1 \mid \mathrm{Q}]:\varphi$

iff $(\forall \mu \in \mathcal{N}_\rho(n) : \mu \in \gamma(*)\ \wedge$

$\forall v_1,\cdots,v_k\colon \wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i)$

$\Rightarrow <v_1,\cdots,v_k>_{\varepsilon}[\mathrm{dest}\ \mathcal{D}] \subseteq \kappa(\mu)) \wedge$

$(\gamma,\kappa,\rho) \vDash^* n[(M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\circ}.P_1 \mid \mathrm{Q}]:\varphi$

iff $(\forall \mu \in \mathcal{N}_\rho(n) : \mu \in \gamma(*)\ \wedge$

$\forall v_1,\cdots,v_k\colon \wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i)$

$\Rightarrow <v_1,\cdots,v_k>_{\varepsilon}[\mathrm{dest}\ \mathcal{D}] \subseteq \kappa(\mu)) \wedge$

$\forall \mu' \in \mathcal{N}_\rho(n) : \mu' \in \gamma(*) \wedge (\gamma,\kappa,\rho) \vDash^{\mu'} (M'_1,\cdots,M'_j;x_{j+1},\cdots,x_k)^{\circ}.P_1 \mid \mathrm{Q}:\varphi$

iff $(\forall \mu \in \mathcal{N}_\rho(n) : \mu \in \gamma(*) \wedge$

$\forall v_1, \cdots, v_k\colon \wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i)$

$\Rightarrow <v_1, \cdots, v_k>_\varepsilon [\text{dest } \mathcal{D}] \subseteq \kappa(\mu)) \wedge$

$\forall \mu' \in \mathcal{N}_\rho(n) : \mu' \in \gamma(*) \wedge (\gamma,\kappa,\rho) \models^{\mu'} (M_1', \cdots, M_j'; x_{j+1}, \cdots, x_k)^{\circ}.P_1 \wedge (\gamma,\kappa,\rho) \models^{\mu'} \mathrm{Q} : \varphi$

iff $(\forall \mu \in \mathcal{N}_\rho(n) : \mu \in \gamma(*) \wedge$

$\forall v_1, \cdots, v_k\colon \wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i)$

$\Rightarrow <v_1, \cdots, v_k>_\varepsilon [\text{dest } \mathcal{D}] \subseteq \kappa(\mu)) \wedge$

$\forall \mu' \in \mathcal{N}_\rho(n) : \mu' \in \gamma(*) \wedge$

$(\forall <v_1, \cdots, v_k>_\ell [\text{dest } \mathcal{L}] \in \kappa(\mu') \wedge$

$\wedge_{i=1}^{j} v_i \in \mathcal{M}_\rho(M_i')$

$\Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(\lfloor x_i \rfloor) \wedge$

$(\neg \mathrm{RM}(\ell, \mathcal{D}, \varepsilon, \mathcal{L}) \Rightarrow (\ell, \varepsilon) \in \varphi) \wedge (\gamma,\kappa,\rho) \models^{\mu'} P_1 : \varphi) \wedge (\gamma,\kappa,\rho) \models^{\mu'} \mathrm{Q} : \varphi$

As $\mathcal{N}_\rho(n) = \{\lfloor n \rfloor\}$, the above analysis amounts to:

$(\lfloor n \rfloor \in \gamma(*) \wedge \forall v_1, \cdots, v_k\colon \wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i)$ (1)

$\Rightarrow <v_1, \cdots, v_k>_\varepsilon [\text{dest } \mathcal{D}] \subseteq \kappa(\lfloor n \rfloor)) \wedge$

$\lfloor n \rfloor \in \gamma(*) \wedge$ (2)

$(\forall <v_1, \cdots, v_k>_\ell [\text{dest } \mathcal{L}] \in \kappa(\lfloor n \rfloor) \wedge$

$\wedge_{i=1}^{j} v_i \in \mathcal{M}_\rho(M_i')$

$\Rightarrow \wedge_{i=j+1}^{k} v_i \in \rho(\lfloor x_i \rfloor) \wedge$ (3)

$(\neg \mathrm{RM}(\ell, \mathcal{D}, \varepsilon, \mathcal{L}) \Rightarrow (\ell, \varepsilon) \in \varphi) \wedge$

$(\gamma,\kappa,\rho) \models^{n} P_1) \wedge (\gamma,\kappa,\rho) \models^{n} \mathrm{Q} : \varphi$ (4)

Since by fv(M_i) = $\emptyset$ and (2), then (1) gives $<\lfloor M_1 \rfloor, \cdots, \lfloor M_k \rfloor>_\varepsilon [\text{dest } \mathcal{D}] \in \kappa(\lfloor n \rfloor)$. The assumption $\wedge_{i=1}^{j} M_i = M_i'$ gives $\wedge_{i=1}^{j} \lfloor M_i \rfloor \in \mathcal{M}_\rho(M_i')$. Thus from (3) and (4) we have $\wedge_{i=j+1}^{k} \lfloor M_i \rfloor \in \rho(\lfloor x_i \rfloor)$ and $(\gamma,\kappa,\rho) \models^{n} P_1 : \varphi$. The Lemma 1 then gives $(\gamma,\kappa,\rho) \models^{n} P_1\{x_1 \leftarrow M_1\} \cdots \{x_k \leftarrow M_k\}{:}\varphi$. The analysis definition of ambient and parallelism then allows to conclude that $(\gamma,\kappa,\rho) \models^{*} n[P_1\{x_1 \leftarrow M_1\} \cdots \{x_k \leftarrow M_k\} \mid \mathrm{Q}] : \varphi$ i.e. that $(\gamma,\kappa,\rho) \models^{*} P' : \varphi$ as required. The proof of the second part of the result is similar to that of the case (Output-Chd1).

Case (Output-Chd2), **(Output-Par1)** and **(Output-Par2)** are similar.

This completes the whole proof.

□

D.2 Hardest Attacker

Theorem 4: Soundness of adjusted Dolev-Yao attacker

If $(\gamma,\kappa,\rho,\varphi)$ satisfies $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ of type $(\mathcal{N}_{\mathrm{f}},\mathcal{A}_{\mathcal{K}})$ then $(\gamma,\kappa,\rho)\models^{*}\overline{Q}:\varphi$ for all well-formed processes $\overline{Q}$ of type $(\mathcal{N}_{\mathrm{f}},\mathcal{A}_{\mathcal{K}})$.

Proof. A process $\overline{Q}$ has extended type $(\{z_{\bullet}, t_{\bullet}\}, \mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet}, n_{\star}\}, \mathcal{A}_{\mathcal{K}})$ whenever the canonical variables are in $\{z_{\bullet}, t_{\bullet}\}$, the canonical names are in $\lfloor\mathcal{N}_{\mathrm{f}}\rfloor\cup\lfloor\mathcal{N}_{\mathrm{v}}\rfloor\cup\{n_{\bullet},n_{\star}\}$, all the arities of input and output are in $\mathcal{A}_{\mathcal{K}}$. By structural induction on $\overline{\mathrm{Q}}$ we prove:

If $(\gamma,\kappa,\rho,\varphi)$ satisfies $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ of type $(\mathcal{N}_{\mathrm{f}},\mathcal{A}_{\mathcal{K}})$ then $(\gamma,\kappa,\rho)\models^{*}\overline{Q}:\varphi$ for all processes $\overline{Q}$ of extended type $(\{z_{\bullet}, t_{\bullet}\}, \mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet},n_{\star}\},\mathcal{A}_{\mathcal{K}})$, where $\overline{Q}$ is any well-formed site ambients, $\Gamma\vdash wf_{S}(\overline{Q})$.

In below cases, we suppose $\overline{Q}$ is enclosed by ambient *.

Case $\overline{Q}=(\nu\ n)P$. We first show that the continuous process P has the extended type $(\{z_{\bullet}, t_{\bullet}\}, (\mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet},n_{\star}\}),\mathcal{A}_{\mathcal{K}})$. By assumption $\lfloor m\rfloor$ is coalesced to be $n_{\bullet}$ for all $m\in(\mathrm{fn}(\overline{Q})\backslash(\mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet},n_{\star}\}))\cup\mathrm{bn}(\overline{Q})$. This requirement is weakened for P since $\lfloor m'\rfloor$ is required be $n_{\bullet}$ only for the subset:

$$\begin{aligned}&(\mathrm{fn}(P)\backslash(\mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet},n_{\star}\}))\cup\mathrm{bn}(P)\\ =&((\mathrm{fn}(\overline{Q})\cup\{m\})\backslash(\mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet},n_{\star}\}))\cup(\mathrm{bn}(\overline{Q})\setminus\{m\})\\ \in\ &(\mathrm{fn}(\overline{Q})\backslash(\mathcal{N}_{\mathrm{f}}\cup\mathcal{N}_{\mathrm{v}}\cup\{n_{\bullet},n_{\star}\}))\cup\mathrm{bn}(\overline{Q})\end{aligned}$$

This shows P has the desired extended type. Then the induction hypothesis and the analysis allow us to conclude that $(\gamma,\kappa,\rho)\models^{*}\overline{Q}:\varphi$.

Case $\overline{Q}=(\nu k\ n)P$ is similar.

Case $\overline{Q}=!\overline{P}$ and $\overline{Q}=\overline{P_1}\,|\,\overline{P_2}$ hold according to the induction hypothesis and the definition of the analysis.

Case $\overline{Q}=0$ holds trivially by the analysis $(\gamma,\kappa,\rho)\models^{*}0:\varphi$.

Case $\overline{Q}=<M_1, \ldots, M_k>^{\uparrow}$. First note that $k\in\mathcal{A}_{\mathcal{K}}$ so the length of the output is covered by $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$. Suppose the parent of current ambient * is ambient μ_p. Because ambient μ_p encloses ambient * which then encloses $\overline{Q}$, then whichever μ_p a free name or a restricted name we always have $\mu_p \in \mathcal{N}_{\mathrm{f}} \cup \mathcal{N}_{\mathrm{v}}$ and thus $\mu_p \in \rho(z_\bullet)$ according to the component 5 of $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$. Then $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ gives that $\forall v_1,\cdots,v_k$: $\wedge_{i=1}^{k} v_i \in \rho(z_\bullet) \Rightarrow <v_1,\cdots,v_k >_\varepsilon$ [dest $\mathcal{D}$] $\in \kappa(\mu_p)$.

To establish $(\gamma,\kappa,\rho) \models^* < M_1,...,M_k >^{\uparrow} : \varphi$ according to the definition of the analysis, w need prove $\mathcal{M}_\rho(M_i) \subseteq \rho(z_\bullet)$. If M_i, is a free variable, intuitively it can be any value v_i such that $v_i \in \rho(z_\bullet)$. The proof relies on the fact that M_i is not an arbitrary capability but appears in $\rho(z_\bullet)$ of $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ attacker with extended type ({ $z_\bullet$, $t_\bullet$ }, ($\mathcal{N}_{\mathrm{f}} \cup \mathcal{N}_{\mathrm{v}}$ $\cup$ { $n_\bullet$, $n_\bigstar$ }), $\mathcal{A}_{\mathcal{K}}$). This allows us to conclude that $\forall v_1,\cdots,v_k$: $\wedge_{i=1}^{k} v_i \in \mathcal{M}_\rho(M_i) \Rightarrow <v_1,\cdots,v_k >_\varepsilon$ [dest $\mathcal{D}$] $\subseteq \kappa(\mu_p)$.

Case $\overline{Q}$=<*M*₁, …, *M*ₖ>$^{\circ}_{\varepsilon}$[dest $\mathcal{D}$]. Similar to the above case. Please also note the message of the output is not supposed to be encrypted as it is not enclosed by a packet.

Case $\overline{Q}$ =<*M*₁, …, *M*ₖ>^{n} . Note that the fact that $\lfloor n \rfloor$ appears in $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ having extended type ({ $z_\bullet$, $t_\bullet$ }, ($\mathcal{N}_{\mathrm{f}} \cup \mathcal{N}_{\mathrm{v}}$ $\cup$ { $n_\bullet$, $n_\bigstar$ }), $\mathcal{A}_{\mathcal{K}}$). Assume the ambient *n* is enclosed by ambient *, then the analysis gives that $\lfloor n \rfloor \in \gamma(*)$. Then $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ ensures that $\forall v_1,\cdots,v_k$: $\wedge_{i=1}^{k} v_i \in \rho(z_\bullet) \Rightarrow <v_1,\cdots,v_k >_\varepsilon$ [dest $\mathcal{D}$] $\in \kappa(\lfloor n \rfloor)$. The left part of proof is similar to the above two cases.

Case $\overline{Q} = (M_1,\cdots,M_j; x_{j+1},\cdots,x_k)^{\circ}.\overline{P}$. $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ gives that $\forall < v_1,...,v_k >_\varepsilon$ [dest $\mathcal{D}$] $\in \kappa(*)$: $\wedge_{i=1}^{k} v_i \in \rho(z_\bullet)$.

Next if M_i is a free variable, it can be any value v_i such that $v_i \in \rho(z_\bullet)$. Since by the fact that M_i is not an arbitrary capability but appears in $\rho(z_\bullet)$ of the adjusted Dolev-Yao attacker with extended type ({ $z_\bullet$, $t_\bullet$ }, ($\mathcal{N}_{\mathrm{f}} \cup \mathcal{N}_{\mathrm{v}}$ $\cup$ { $n_\bullet$, $n_\bigstar$ }), $\mathcal{A}_{\mathcal{K}}$), we have that $\mathcal{M}_\rho(M_i) \subseteq \rho(z_\bullet)$. For v_i' such that $\forall v_1',\cdots,v_j'$: $\wedge_{i=1}^{j} v_i' \in \mathcal{M}_\rho(M_i)$, assume there are $\wedge_{i=1}^{j} v_i' = v_i$, we then have $\wedge_{i=j+1}^{k} v_i \in \rho(z_\bullet)$ since $\lfloor x_i \rfloor = z_\bullet$. Last the induction hypothesis takes care of the continuous process $\overline{P}$, which has same extended type as $\overline{Q}$, and this completes the proof for the case.

Case $\overline{Q} = (M_1,\cdots,M_j; x_{j+1},\cdots,x_k)^{n}_{\ell_\bullet}$[orig $\mathcal{D}$].$\overline{P}$. First note that $\lfloor n \rfloor$ appears in $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ that is $\lfloor n \rfloor \in \rho(z_\bullet)$. Then assume there is an ambient *n* showing up inside ambient *, the analysis then gives us that $\lfloor n \rfloor \in \gamma(*)$. Since by $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ we have that $\forall < v_1,...,v_k >_\ell$ [dest $\mathcal{L}$] $\in \kappa(\lfloor n \rfloor)$: $\wedge_{i=1}^{k} v_i \in \rho(z_\bullet)$ and additionally $\neg$RM$(\ell,\mathcal{D},\ell_\bullet,\mathcal{L}) \Rightarrow$ $(\ell,\ell_\bullet) \in \varphi$.

The left part of the proof is very similar to the case of local input presented above except that this time we also consider that $\neg\mathrm{RM}(\ell,\mathcal{D},\ell_{\bullet},\mathcal{L}) \Rightarrow (\ell,\ell_{\bullet}) \in \varphi$ which also has been given by $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$ as above.

Case $\overline{\mathrm{Q}} = N[\,\overline{P}\,]$ for $\mathcal{N}_{\rho}(N) \subseteq \lfloor \mathcal{C}_{p} \rfloor$. Here $\mathcal{N}_{\rho}(N) \subseteq \rho(z_{\bullet})$ holds since $\overline{\mathrm{Q}}$ has the extended type $(\{z_{\bullet},\ t_{\bullet}\},\ (\mathcal{N}_{\mathrm{f}} \cup \mathcal{N}_{\mathrm{v}} \cup \{n_{\bullet}, n_{\bigstar}\}), \mathcal{A}_{\mathcal{K}})$. However, this time we can not apply the induction hypothesis to sub-process $\overline{P}$ because it follows the rule defined by the well-formedness $\Gamma \vdash wf_{\mathrm{p}}(P_{\bigstar})$. This amounts to inductively prove on the process $\overline{P}$ following the well-formedness $\Gamma \vdash wf_{\mathrm{p}}(P_{\bigstar})$. In particular by the Lemma 4 of Chapter 4, the case that $\overline{P}$ is an ambient is skipped. The most interesting cases are when the considered process $\overline{P}$ are in $N'.\overline{P'}$ or out $N'.\overline{P'}$. We need prove that the packet N may move into or out of N'. We first have $\mathcal{N}_{\rho}(N') \subseteq \rho(z_{\bullet})$ as $\overline{P}$ has the same extended type as $\overline{\mathrm{Q}}$. Intuitively any N', which the packet N may move in or out, must be reachable for the ambient *. This is also covered by $\mathcal{F}_{\mathrm{RM}}^{\mathrm{A\_DY}}$. Since $\overline{P'}$ follows the extended type, the induction hypothesis then applies on $\overline{P'}$.

Last by Assumption 1, we can skip the case that $\overline{\mathrm{Q}} = N[\,\overline{P}\,]$ for $\mathcal{N}_{\rho}(N) \subseteq \lfloor \mathcal{C}_{s} \rfloor$. This completes the proof.

□

Appendix E Source Code of ABoxed Ambients Tool

E.1 mlba.sml

```
structure MLBA =

struct

   (* Meta level language *)
   type Index   = string
            type Group = string * Index list
            type CP    = string * Index list
            type Lab   = string * Index list
            type Set     = Index list

            datatype Naming = Name of string * Index list
                  | Var of string * Index list

            datatype Capability = In  of Naming
                                          | Out of Naming
                                          | NX  of Naming

            datatype Process = New of Naming * Group * Process
                  | INew      of (Index * Set) list * Naming * Group * Process
                  | News      of Naming * Group * Process
                  | INews     of (Index * Set) list * Naming * Group * Process
                  | Newp      of Naming * Group * Process
                  | INewp     of (Index * Set) list * Naming * Group * Process
                  | Nil
                  | Par        of Process * Process
                  | IPar       of Index * Set * Process
                  | Rep        of Process
                  | Amb       of Naming * Process
                  | Pre        of Capability * Process
                  | Sendc     of Capability list * Naming
                  | Sendl     of Capability list * CP * CP list * Lab
                  | Sendp     of Capability list
                  | Receivec  of Capability list * Naming list * Naming * CP * CP list * Lab * Process
                  | Receivel  of Capability list * Naming list * Process
                  | Receivep  of Capability list * Naming list * Process
                  | Attacker


val NATURAL1  = ["1"]
val NATURAL2  = ["1","2"]
val NATURAL3  = ["1","2","3"]

local

            fun substIndexIndex (i,a) (i1)      = if not(i1=i) then i1 else a

   fun substIndexCP (i,a) (c,il)       = (c,List.map (substIndexIndex (i,a)) il)

   fun substIndexLab (i,a) (l,il)      = (l,List.map (substIndexIndex (i,a)) il)

   fun substIndexGroup (i,a) (g,il)    = (g,List.map (substIndexIndex (i,a)) il)

   fun substIndex (i,a) (Name(n,il))   =
        Name(n,List.map (substIndexIndex (i,a)) il)
      | substIndex (i,a) (Var(x,il))  =
        Var(x,List.map (substIndexIndex (i,a)) il)
```

```
  fun substIndexCap (i,a) (In(n))     = In(substIndex (i,a) n)
    | substIndexCap (i,a) (Out(n))    = Out(substIndex (i,a) n)
    | substIndexCap (i,a) (NX(n))     = NX(substIndex (i,a) n)

  fun substIndexList (i,a) []         = []
    | substIndexList (i,a) (N::NL)    = (substIndex (i,a) (N))::(substIndexList (i,a) (NL))

  fun substIndexCPList (i,a) []       = []
    | substIndexCPList (i,a) (C::CL)  = (substIndexCP (i,a) (C))::(substIndexCPList (i,a) (CL))

  fun substIndexCapList (i,a) []      = []
    | substIndexCapList (i,a) (M::ML) = (substIndexCap (i,a) (M))::(substIndexCapList (i,a) (ML))

  fun formList ([],Name(n1,il1),(n2,il2))           = [(Name(n1,il1),(n2,il2))]
    | formList ((i,[])::nl,Name(n1,il1),(n2,il2))    = []
    | formList ((i,h::il)::nl,Name(n1,il1),(n2,il2)) =
        formList(nl,substIndex (i,h) (Name(n1,il1)),substIndexGroup (i,h) (n2,il2))@
        formList((i,il)::nl,Name(n1,il1),(n2,il2))
    | formList _                                 = []


  fun substProcess ((i,h),New(N,gn,P))     =
        New(substIndex (i,h) N,substIndexGroup (i,h) gn,substProcess((i,h),P))
    | substProcess ((i,h),INew(il,N,gn,P)) =
        INew(il,N,gn,substProcess((i,h),P))
    | substProcess ((i,h),News(N,gn,P))    =
        News(substIndex (i,h) N,substIndexGroup (i,h) gn,substProcess((i,h),P))
    | substProcess ((i,h),INews(il,N,gn,P))=
        INews(il,N,gn,substProcess((i,h),P))
    | substProcess ((i,h),Newp(N,gn,P))    =
        Newp(substIndex (i,h) N,substIndexGroup (i,h) gn,substProcess((i,h),P))
    | substProcess ((i,h),INewp(il,N,gn,P))=
        INewp(il,N,gn,substProcess((i,h),P))
    | substProcess ((i,h),Nil)           = Nil
    | substProcess ((i,h),Par(P1,P2))     = Par(substProcess((i,h),P1),substProcess((i,h),P2))
    | substProcess ((i,h),IPar(i',il,P))  = IPar(i',il,substProcess((i,h),P))
    | substProcess ((i,h),Rep(P))         = Rep(substProcess((i,h),P))
    | substProcess ((i,h),Amb(N,P))       = Amb(substIndex (i,h) N,substProcess((i,h),P))
    | substProcess ((i,h),Pre(M,P))       = Pre(substIndexCap (i,h) M,substProcess((i,h),P))
    | substProcess ((i,h),Sendc(ML,N))    = Sendc(substIndexCapList (i,h) ML,substIndex (i,h) N)
    | substProcess ((i,h),Sendl(ML,C,CL,L)) =
        Sendl(substIndexCapList (i,h) ML,substIndexCP (i,h) C,substIndexCPList (i,h) CL,
            substIndexLab (i,h) L)
    | substProcess ((i,h),Sendp(ML))      = Sendp(substIndexCapList (i,h) ML)
    | substProcess ((i,h),Receivec(ML,NL,K,C,CL,L,P)) =
        Receivec(substIndexCapList (i,h) ML,substIndexList (i,h) NL,substIndex (i,h) K,
              substIndexCP (i,h) C,substIndexCPList (i,h) CL,substIndexLab (i,h) L,
              substProcess((i,h),P))
    | substProcess ((i,h),Receivel(ML,NL,P)) =
        Receivel(substIndexCapList (i,h) ML,substIndexList (i,h) NL,substProcess((i,h),P))
    | substProcess ((i,h),Receivep(ML,NL,P)) =
        Receivep(substIndexCapList (i,h) ML,substIndexList (i,h) NL,substProcess((i,h),P))
    | substProcess ((i,h),Attacker)        = Attacker

  fun formProcList (i,[],P)      = []
    | formProcList (i,h::il,P)   =
        substProcess((i,h),P)::formProcList(i,il,P)


in
   fun instNew (INew(nl,Name(n1,il1),(n2,il2),P)) =
              let val ngs = formList (nl,Name(n1,il1),(n2,il2))
                  fun inst []          = P
                    | inst ((n,g)::ngl) = New(n,g,inst ngl)
              in
                 inst ngs
              end
     | instNew (INews(nl,Name(n1,il1),(n2,il2),P)) =
            let val ngs = formList (nl,Name(n1,il1),(n2,il2))
                fun inst []         = P
                  | inst ((n,g)::ngl) = News(n,g,inst ngl)
            in
               inst ngs
            end
  | instNew (INewp(nl,Name(n1,il1),(n2,il2),P)) =
            let val ngs = formList (nl,Name(n1,il1),(n2,il2))
                fun inst []         = P
                  | inst ((n,g)::ngl) = Newp(n,g,inst ngl)
            in
               inst ngs
            end
  | instNew _                              = Nil


fun instProcess (IPar(n,il,P)) =
     let val ps = formProcList (n,il,P)
         fun inst []      = Nil
           | inst (P1::Pl) = Par(P1,inst Pl)
     in
         inst ps
     end
  | instProcess _           = Nil

(* Functions converting processes and its subparts into strings *)
(* Used both for pretty printing and clause generation          *)
fun toStringIndexList []      = ""
  | toStringIndexList [I]     = I
  | toStringIndexList (I::IL)  = I^toStringIndexList(IL)

fun toStringNaming (Name(n,il))   = n^toStringIndexList(il)
  | toStringNaming (Var(x,il))    = x^toStringIndexList(il)

fun toStringNamingList []      = ""
  | toStringNamingList [N]     = toStringNaming(N)
  | toStringNamingList (N::NL) = toStringNaming(N)^","^toStringNamingList(NL)

fun toStringCapability (In(N))  = "in("^toStringNaming(N)^")"
  | toStringCapability (Out(N)) = "out("^toStringNaming(N)^")"
  | toStringCapability (NX(N))  = toStringNaming(N)

fun toStringCapaList []        = ""
  | toStringCapaList [M]       = toStringCapability(M)
  | toStringCapaList (M::ML)   = toStringCapability(M)^","^toStringCapaList(ML)

fun toStringCP (cp,[])         = cp
  | toStringCP (cp,il)         = cp^toStringIndexList(il)

fun toStringCPList []          = ""
  | toStringCPList [C]         = toStringCP(C)
  | toStringCPList (C::CL)     = toStringCP(C)^","^toStringCPList(CL)

fun toStringGroup (gn,[])      = gn
  | toStringGroup (gn,il)      = gn^toStringIndexList(il)

fun toStringLab (ln,[])        = ln
  | toStringLab (ln,il)        = ln^toStringIndexList(il)


fun toStringINewIndex []       = ""
  | toStringINewIndex [(i,il)]  = i^" in {"^toStringIndexList(il)^"}"
  | toStringINewIndex ((i,il)::nl) = i^" in {"^toStringIndexList(il)^"}, "^toStringINewIndex(nl)

fun toStringProcess (New(N,gn,P))   = "(new "^toStringNaming(N)^": "^
     toStringGroup(gn)^") \n"^toStringProcess(P)
  | toStringProcess (INew(il,N,gn,P))=
     "(new_{"^toStringINewIndex(il)^"} "^toStringNaming(N)^": "^
     toStringGroup(gn)^") \n"^toStringProcess(P)
```

```
    | toStringProcess (News(N,gn,P))   = "(newsite "^toStringNaming(N)^
        ": "^toStringGroup(gn)^") \n"^toStringProcess(P)
    | toStringProcess (INews(il,N,gn,P))=
        "(newsite_{"^toStringINewIndex(il)^"} "^toStringNaming(N)^": "^
        toStringGroup(gn)^") \n"^toStringProcess(P)
    | toStringProcess (Newp(N,gn,P))   = "(newpriv "^toStringNaming(N)^
        ": "^toStringGroup(gn)^") \n"^toStringProcess(P)
    | toStringProcess (INewp(il,N,gn,P))=
        "(newpriv_{"^toStringINewIndex(il)^"} "^toStringNaming(N)^": "^
        toStringGroup(gn)^") \n"^toStringProcess(P)
    | toStringProcess (Nil)           = "0"
    | toStringProcess (Par(P1,P2))    = toStringProcess(P1) ^ "\n|\n" ^ toStringProcess(P2)
    | toStringProcess (IPar(i,il,P))  =
        "(|_"^i^" {"^toStringIndexList(il)^"} \n"^toStringProcess(P)^" \n)"
    | toStringProcess (Rep(P))        = "! " ^ toStringProcess(P)
    | toStringProcess (Amb(N,P))      = toStringNaming(N) ^ "[ " ^ toStringProcess(P) ^ " ]"
    | toStringProcess (Pre(M,P))      = toStringCapability(M) ^ "." ^ toStringProcess(P)
    | toStringProcess (Sendc(ML,N))   = "< " ^
        toStringCapaList(ML)^
        ">{"^toStringNaming(N)^"}"
    | toStringProcess (Sendl(ML,C,CL,_)) = "< " ^
        toStringCapaList(ML)^
        ">{o}[at "^toStringCP(C)^" dest {"^
        toStringCPList(CL)^
        "}]"
    | toStringProcess (Sendp(ML))     = "< "^
        toStringCapaList(ML)^
        ">{^}"
    | toStringProcess (Receivec(ML,NL,K,C,CL,_,P)) =
        "( "^
        toStringCapaList(ML)^
        "; " ^
        toStringNamingList(NL)^
        "){"^toStringNaming(K)^
        "}[at "^toStringCP(C)^" orig {"^toStringCPList(CL)^"}].\n"^
        toStringProcess(P)
    | toStringProcess (Receivel(ML,NL,P)) = "( "^
        toStringCapaList(ML)^
        "; " ^
        toStringNamingList(NL)^
        "){o}.\n"^toStringProcess(P)
    | toStringProcess (Receivep(ML,NL,P)) = "( "^
        toStringCapaList(ML)^
        "; " ^
        toStringNamingList(NL)^
        "){^}.\n"^toStringProcess(P)
    | toStringProcess (Attacker)      = "Attacker"


  fun toString P = toStringProcess P

  fun print msg =
      let val fi = "./test/testml"
          val fh = TextIO.openOut(fi)
      in
          TextIO.output(fh, msg);
          TextIO.closeOut(fh)
      end
end
end
```

E.2 analysis.sml

```
structure ANALYSIS =

struct
   exception new_expects_a_name
   exception no_attacked_program
   exception unclear_priv_type

local

             open MLBA

              fun print msg = TextIO.output(TextIO.stdOut, msg);

              fun lookup G n =
                   let fun h []          = (print ("lookup "^n); raise Fail "unknown variable")
                        | h ((y,v,_)::yr) = if n=y then v else h yr
                   in h G end

   (* Auxiliary functions for generating clauses *)

   val nextmu = ref 0

   fun next () = let val _ = nextmu := !nextmu+1  in  !nextmu end

              (* Reachability predicate *)
   val rc = ref 0
   fun newReachCount() = (rc:= !rc+1;"RC"^Int.toString(!rc))

   fun quantify(l,u,Q,x) =
                   let
                      val s =  Q^" "^x^"_"^(Int.toString l)^". "
                   in
                      if l > u then ""
                      else s^quantify(l+1,u,Q,x)
                   end

   fun queryVar(l,u,x) =
        let
              val s = x^"_"^(Int.toString l)
        in
              if l > u then ""
              else if l = u then s
              else s^", "^queryVar(l+1,u,x)
        end


   fun queryRelList(l,R,x,[]) = "1"
     | queryRelList(l,R,x,h::ls) = R^"("^h^", "^x^"_"^Int.toString(l)^") & "^
                                                 queryRelList(l+1,R,x,ls)

              fun labelList (l,[])      = "1"
                        | labelList (l,H1::LL) =
                           let val H1' = toStringLab H1
                           in  " D("^l^", "^H1'^") &"^labelList(l,LL)
                           end

              fun cleanList []    = []
     | cleanList (h::tl) = h::(cleanList(List.filter (fn t => t <> h) tl))

   fun collectRecArity (a,New(Name(_),_,P))         = collectRecArity (a,P)
     | collectRecArity (a,New(Var(_),_,_))         = raise new_expects_a_name
     | collectRecArity (a,INew(_,_,_,P))          = collectRecArity (a,P)
     | collectRecArity (a,News(Name(_),_,P))        = collectRecArity (a,P)
     | collectRecArity (a,News(Var(_),_,_))        = raise new_expects_a_name
     | collectRecArity (a,INews(_,_,_,P))          = collectRecArity (a,P)
     | collectRecArity (a,Newp(Name(_),_,P))        = collectRecArity (a,P)
```

```
  | collectRecArity (a,Newp(Var(_),_,_))        = raise new_expects_a_name
  | collectRecArity (a,INewp(_,_,_,P))          = collectRecArity (a,P)
  | collectRecArity (a,Nil)                     = a
  | collectRecArity (a,Par(P1,P2))              = collectRecArity(a,P1)@collectRecArity([],P2)
  | collectRecArity (a,IPar(_,_,P))             = collectRecArity(a,P)
  | collectRecArity (a,Rep(P))                  = collectRecArity (a,P)
  | collectRecArity (a,Amb(_,P))                = collectRecArity (a,P)
  | collectRecArity (a,Pre(_,P))                = collectRecArity (a,P)
  | collectRecArity (a,Sendc(ML,_))             =
     let val k = List.length ML
     in  k::a
     end
  | collectRecArity (a,Receivec(ML,XL,_,_,_,_,P))=
     let val k = List.length ML
        val l = (List.length XL) + k
     in  collectRecArity(l::a,P)
     end
  | collectRecArity (a,Sendl(ML,_,_,_))         =
     let val k = List.length ML
     in  k::a
     end
  | collectRecArity (a,Receivel(ML,XL,P))       =
     let val k = List.length ML
        val l = (List.length XL) + k
     in  collectRecArity(l::a,P)
     end
  | collectRecArity (a,Sendp(ML))               =
     let val k = List.length ML
     in  k::a
     end
  | collectRecArity (a,Receivep(ML,XL,P))       =
     let val k = List.length ML
        val l = (List.length XL) + k
     in  collectRecArity(l::a,P)
     end
  | collectRecArity (a,Attacker)                = a


(* functions for clause generation *)

fun genAttProcess(s,G,P) =
     let val al' = collectRecArity ([],P)
        val al  = cleanList al'
        fun genRZ([])        = " 1 \n"
          | genRZ((n,mu,"y")::tl) = " RZ("^mu^") & "^"RZ(in("^mu^")) & "^
              "RZ(out("^mu^")) & \n"^genRZ(tl)
          | genRZ((n,mu,"n")::tl) = genRZ(tl)
          | genRZ((_,_,_)::tl)    = raise unclear_priv_type

        fun genRZList(l,u,x) =
           let val t = x^"_"^Int.toString(l)
              val s = "RZ("^t^") & \n"^
                   "(A m. "^t^"=in(m) => RZ(out(m)) & RZ(m)) & \n"^
                   "(A m. "^t^"=out(m) => RZ(in(m)) & RZ(m)) & \n"^
                   "(A m. Group("^t^") => RZ(in("^t^")) & RZ(out("^t^"))) "
           in if l > u then "1 \n"
                   else if l = u then s
                   else s^"& \n"^genRZList(l+1,u,x)
           end

        fun queryRZList(l,u,x) =
           let val s = "RZ("^x^"_"^Int.toString(l)^")"
           in if l>u then "1 \n"
             else if l=u then s
             else s^"& \n"^queryRZList(l+1,u,x)
           end

        fun genEavesdrop1([],s)    = "1 \n"
          | genEavesdrop1(k::tl,s) = "("^quantify(1,k,"A","t")^
        "C"^Int.toString(k)^"("^s^", "^queryVar(1,k,"t")^", EPSI, ANYWHERE) => \n"^
        genRZList(1,k,"t")^" \n"^
        ") &\n"^genEavesdrop1(tl,s)

    fun genEavesdrop2([],s)    = "1 \n"
      | genEavesdrop2(k::tl,s) = "("^quantify(1,k,"A","t")^"A mu. A ELLO. A LOS."^
        "I("^s^",mu) & \n"^
        "Pack(mu) & \n"^
        "C"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", ELLO, LOS) =>  \n\t"^
        "RT"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", ELLO, LOS) \n"^
        ") &\n"^genEavesdrop2(tl,s)

    fun genDecMsg([])          = "1 \n"
      | genDecMsg(k::tl)       = "(A mu. A ELLO. A LOS."^
        quantify(1,k,"A","t")^" \n\t"^
        "RT"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", ELLO, LOS) & \n\t\t"^
        "RZ(mu) => ("^genRZList(1,k,"t")^" &\n\t\t"^
        "(LOS != ANYWHERE => FI(ELLO,DYA)) \n"^
                "")) &\n"^genDecMsg(tl)


            fun genEncMsg([])          = "1 \n"
              | genEncMsg(k::tl)       = "(A mu."^quantify(1,k,"A","t")^" \n\t"^
                "RZ(mu) & Group(mu) & Pack(mu) & "^queryRZList(1,k,"t")^" => \n"^
                "RT"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", DYA, ANYWHERE) \n"^
                ") &\n"^genEncMsg(tl)

    fun genPlainMsg([],s)       = "1 \n"
      | genPlainMsg(k::tl,s)    = "("^quantify(1,k,"A","t")^" \n\t"^
        queryRZList(1,k,"t")^" => \n"^
        "(A mu.ReachSite("^s^", mu) =>\n"^
        "RT"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", EPSI, ANYWHERE) \n"^
                "")) &\n"^genPlainMsg(tl,s)

            fun sendMsg([])            = "1 \n"
              | sendMsg(k::tl)         = "("^quantify(1,k,"A","t")^"A mu.A ELLO.A LOS. \n\t"^
                "RT"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", ELLO, LOS) => \n"^
                "C"^Int.toString(k)^"(mu, "^queryVar(1,k,"t")^", ELLO, LOS) \n"^
                ") &\n"^sendMsg(tl)

in
    (* initial knowledge of an attacker *)
    "\n/* initial knowledge of an attacker */\n"^
    genRZ(G)^" & \n"^
    "Site(TOP) & RZ(TOP) &\n"^

    (* eavesdropping message transferred on internet *)
    "\n/* eavesdropping message transferred on internet */\n"^
    genEavesdrop1(al,s)^" & \n"^
    genEavesdrop2(al,s)^" & \n"^

    (* decrypt encrypted message *)
    "\n/* decrypt encrypted message */\n"^
    genDecMsg(al)^" & \n"^

    (* compose encryption *)
    "\n/* compose encryption */\n"^
    genEncMsg(al)^" & \n"^

    (* compose plain message *)
    "\n/* compose plain message */\n"^
    genPlainMsg(al,s)^" & \n"^

    (* send message *)
    "\n/* send message */\n"^
    "(A mu.Site(mu) & RZ(mu) => \n"^
    "ReachSite(mu,mu) \n"^
    ") &\n"^
    "(A mu1.A mu2.Site(mu1) & Site(mu2) & RZ(mu1) & RZ(mu2) & I(mu1,mu2) => \n"^
          "ReachSite(mu1,mu2) \n"^
          ") &\n"^
          "(A mu1.A mu2.ReachSite(mu1,mu2) => \n"^
          "ReachSite(mu2,mu1) \n"^
          ") &\n"^
          "(A mu1.A mu2.A mu3.ReachSite(mu1,mu2) & ReachSite(mu2,mu3)=> \n"^
          "ReachSite(mu1,mu3) \n"^
          ") &\n"^
          "(A t.RZ(t) & Group(t) & Pack(t) => \n"^
          "(A mu.ReachSite("^s^",mu) => I(mu,t) \n"^
          ")) & \n"^
          sendMsg(al)
      end

(* auxiliary function N and M *)

fun genN(G,Name(n,il)) =
      let val n' = toStringNaming(Name(n,il))
      in  "N("^n'^", "^(lookup G n')^")"
      end
  | genN(G,Var(x,il))  =
      let val x' = toStringNaming(Var(x,il))
      in  "(A mu. R("^x'^", mu) & Group(mu) => N("^x'^", mu))"
      end

fun genM(G, In(N))       =
      let val n = toStringNaming(N)
      in genN(G,N)^" &\n"^
         "(A mu. A t. N("^n^", mu) => \n \t\t"^  (* Here is diferent from Henrik's code *)
         "M(in("^n^"), in(mu)))"
      end
  | genM(G, Out(N))      =
      let val n = toStringNaming(N)
      in genN(G,N)^" &\n"^
         "(A mu. A t. N("^n^", mu) => \n\t\t"^  (* Here is diferent from Henrik's code *)
         "M(out("^n^"), out(mu)))"
      end
  | genM(G, NX(Name(n,il))) =
      let val n' = toStringNaming(Name(n,il))
      in  "M("^n'^", "^(lookup G n')^")"
      end
  | genM(G, NX(Var(x,il)))  =
      let val x' = toStringNaming(Var(x,il))
      in  "(A t. R("^x'^", t) => M("^x'^", t))"
      end

(* closure condition for in and out capabilities *)

fun phi_in(mu,t) = "(A mua. A mup. \n\t"^
      "I(mua, "^t^") &\n\t"^
      "Pack(mua) &\n\t"^
      "I(mup, mua) &\n\t"^
      "I(mup, "^mu^") =>\n\t\t"^
      "I("^mu^", mua))"

fun phi_out(mu,t) = "(A mua. A mug. \n\t"^
      "I(mua,"^t^") &\n\t"^
      "Pack(mua) &\n\t"^
      "I("^mu^",mua) &\n\t"^
      "I(mug,"^mu^") =>\n\t\t"^
      "I(mug,mua))"

(* clauses for groups are not generated util they are used *)
fun genProcess(s,G,New(Name(n,il),mu,P)) =
      let val n' = toStringNaming(Name(n,il))
          val mu' = toStringGroup(mu)
      in
                    "/* new "^n'^" : "^mu'^" */\n"^
                           "Group("^mu'^") &"^
```

```
                              "Pack("^mu'^") &"^
                              "Cap(in("^mu'^")) & "^
                              "Cap(out("^mu'^")) &\n\n "^
                              genProcess(s,(n',mu',"y")::G,P)
      end
                  | genProcess(s,G,New(Var(_),_,_))   =
                     raise new_expects_a_name
                  | genProcess(s,G,INew(il,Name(n,il'),gn,P))   =
                     let val p' = instNew (INew(il,Name(n,il'),gn,P))
                     in  genProcess(s,G,p')
                     end
                  | genProcess(s,G,INew(il,Var(_),_,_))   =
                     raise new_expects_a_name
                  | genProcess(s,G,News(Name(n,il),mu,P))=
                     let val n' = toStringNaming(Name(n,il))
    val mu' = toStringGroup(mu)
  in
                  "/* newsite "^n'^" : "^mu'^" */\n"^
                              "Group("^mu'^") &"^
                              "Site("^mu'^") &"^
                              "Cap(in("^mu'^")) & "^
                              "Cap(out("^mu'^")) &\n\n "^
                              genProcess(s,(n',mu',"y")::G,P)
      end
                  | genProcess(s,G,News(Var(_),_,_))  =
                     raise new_expects_a_name
                  | genProcess(s,G,INews(il,Name(n,il'),gn,P))   =
                     let val p' = instNew (INews(il,Name(n,il'),gn,P))
                     in  genProcess(s,G,p')
                     end
                  | genProcess(s,G,INews(il,Var(_),_,_))   =
                     raise new_expects_a_name
                  | genProcess(s,G,Newp(Name(n,il),mu,P))=
                     let val n' = toStringNaming(Name(n,il))
    val mu' = toStringGroup(mu)
  in
                  "/* newsite "^n'^" : "^mu'^" */\n"^
                              "Group("^mu'^") &"^
                              "Cap(in("^mu'^")) & "^
                              "Cap(out("^mu'^")) &\n\n "^
                              genProcess(s,(n',mu',"n")::G,P)
      end
                  | genProcess(s,G,Newp(Var(_),_,_))  =
                     raise new_expects_a_name
                  | genProcess(s,G,INewp(il,Name(n,il'),gn,P))   =
                     let val p' = instNew (INewp(il,Name(n,il'),gn,P))
                     in  genProcess(s,G,p')
                     end
                  | genProcess(s,G,INewp(il,Var(_),_,_))   =
                     raise new_expects_a_name
                  | genProcess(s,G,Nil)              =     "/* Nil */ \n1 \n"
                  | genProcess(s,G,Par(P1,P2))       =
                     (case (P1,P2) of
                       (Attacker,Attacker)   => raise no_attacked_program
                     | (_,Attacker)         => genProcess(s,G,P1)^
                                         "\n/* parallel */ \n &\n\n"^
                                         genAttProcess(s,G,P1)^"\n\n"
                     | (Attacker,_)         => genProcess(s,G,P2)^
                                         "\n/* parallel */ \n &\n\n"^
                                         genAttProcess(s,G,P2)^"\n\n"
                     | _                  => genProcess(s,G,P1)^
                         "\n/* parallel */ \n &\n\n"^
                         genProcess(s,G,P2)^"\n\n")
        | genProcess(s,G,IPar(n,il,P))      =
           let val p' = instProcess (IPar(n,il,P))
           in  genProcess(s,G,p')
           end
        | genProcess(s,G,Rep(P))            =
           genProcess(s,G,P)
```

```
            | genProcess(s,G,Amb(N,P))         =
               let val mu = "mu"^Int.toString(next())
                  val n  = toStringNaming(N)
               in
                 "/* ambient "^n^" */\n"^
                 genN(G,N)^" & \n"^
                 "(A "^mu^". N("^n^", "^mu^") => "^
                 "I("^s^", "^mu^") & \n\n"^
                 genProcess(mu,G,P)^
                 ")"
               end
            | genProcess(s,G,Pre(In(N),P))      =
               let val n  = toStringNaming(N)
               in
                 "/* in "^n^" */\n"^
                 genM(G,In(N))^" &\n"^
                 "(A t. M(in("^n^"),t) => I("^s^", t)) &\n"^
                 "(A mu. A t. M(in("^n^"), t) & t = in(mu) =>"^
                 phi_in("mu","t")^
                 ") &\n\n"^
                 genProcess(s,G,P)
               end
            | genProcess(s,G,Pre(Out(N),P))     =
               let val n  = toStringNaming(N)
               in
                 "/* out "^n^" */\n"^
                 genM(G,Out(N))^" &\n"^
                 "(A t. M(out("^n^"),t) => I("^s^", t)) &\n"^
                 "(A mu. A t. M(out("^n^"), t) & t = out(mu) =>"^
                 phi_out("mu","t")^
                 ") &\n\n"^
                 genProcess(s,G,P)
               end
            | genProcess(s,G,Pre(NX(N),P))     =
          let val n = toStringNaming(N)
          in
             "/* prefix "^n^" */\n"^
             genM(G,NX(N))^" &\n"^
             "(A t. M("^n^", t) & Cap(t) => I("^s^", t)) &\n"^
       "(A mu. A t. M("^n^", t) & t= in(mu) =>\n"^
       phi_in("mu","t")^") &\n"^
       "(A mu. A t. M("^n^", t) & t = out(mu) =>\n"^
             phi_out("mu","t")^") & \n\n"^
             genProcess(s,G,P)
          end
   | genProcess(s,G,Sendl(ML,CP,CPL,L))=
      let val k    = List.length ML
         val Ss   = List.map toStringCapability ML
         val CP'  = toStringCP CP
         val L'   = toStringLab L
      in
         "/* Sendl */\n"^
         (String.concat (List.map (fn M => (genM(G,M)^" &\n")) ML))^
         "( "^L'^" != ANYWHERE => "^labelList(L',CPL)^" ) &\n"^
         "("^quantify(1,k,"A","t")^
         queryRelList(1,"M","t",Ss)^" => "^
         "C"^Int.toString(k)^"("^s^", "^
                  queryVar(1,k,"t")^", "^CP'^", "^L'^")) \n\n"

               end
            | genProcess(s,G,Receivel(ML,XL,P))=
               let val k = List.length ML
                  val l = (List.length XL) + k
                  val Ms = List.map toStringCapability ML
                  val Ss = List.map toStringNaming XL
                  val rc =  newReachCount()
               in
                 "/* Receivel */\n"^
                 (String.concat (List.map (fn M => (genM(G,M)^" &\n")) ML))^
```

```
                  "("^quantify(1,l,"A","t")^"A ELLO. A LOS. \n"^
                  "C"^Int.toString(l)^"("^s^", "^queryVar(1,l,"t")^", ELLO, LOS) => "^
                  "("^queryRelList(1,"M","t",Ms)^" => "^
                  queryRelList(k+1,"R","t",Ss)^" & \n"^
                  (* "(LOS != ANYWHERE => PQD(ANYWHERE, OMITABLE, LOS, EPSI)) &\n"^
                  genProcess(s,G,P)^
                  ")) \n\n" *)
                  "(LOS != ANYWHERE => PQD(ANYWHERE, OMITABLE, LOS, EPSI)) &\n"^
                  "REACH("^rc^") \n"^
                  ")) &\n"^
              "(REACH("^rc^") => "^(genProcess(s,G,P))^")"
                end
              | genProcess(s,G,Sendc(ML,N))      =
                 let val k = List.length ML
                     val n = toStringNaming N
                     val Ss = List.map toStringCapability ML
                 in
                     "/* Sendc */\n"^
              genN(G, N)^" &\n"^
              (String.concat (List.map (fn M => (genM(G,M)^" &\n")) ML))^
              "(A mu. N("^n^", mu) & I("^s^", mu) => \n"^
              "("^quantify(1,k,"A","t")^
              queryRelList(1,"M","t",Ss)^
              " => "^
              "C"^Int.toString(k)^"(mu, "^
              queryVar(1,k,"t")^", EPSI, ANYWHERE)))\n\n"
          end
      | genProcess(s,G,Receivec(ML,XL,N,CP,CPL,L,P))=
          let val k   = List.length ML
              val l   = (List.length XL) + k
              val n   = toStringNaming N
              val Ms   = List.map toStringCapability ML
              val Ss   = List.map toStringNaming XL
              val CP'  = toStringCP CP
              val L'   = toStringLab L
              val rc =  newReachCount()
          in
              "/* Receivec */\n"^
              genN(G, N)^" &\n"^
              (String.concat (List.map (fn M => (genM(G,M)^" &\n")) ML))^
                 "("^L'^" != ANYWHERE => "^
                     labelList(L',CPL)^
                     ") & \n"^
              "(A mu. N("^n^", mu) & I("^s^", mu) => \n"^
              "("^quantify(1,l,"A","t")^"A ELLO. A LOS. \n"^
              "C"^Int.toString(l)^"(mu, "^queryVar(1,l,"t")^", ELLO, LOS) & "^
              queryRelList(1,"M","t",Ms)^" => "^
              queryRelList(k+1,"R","t",Ss)^" & \n"^
              "(LOS != ANYWHERE & "^L'^"!= ANYWHERE => PQD("^L'^", ELLO, LOS, "^CP'^"))
& \n"^
              "(LOS = ANYWHERE & ELLO = DYA & "^L'^"!= ANYWHERE => FI(ELLO, "^CP'^"))
& \n"^
              "(LOS = ANYWHERE & ELLO != DYA & "^L'^"!= ANYWHERE =>PQD("^L'^", ELLO,
ANYWHERE, OMITABLE)) & \n"^
              (*"(LOS != ANYWHERE & "^L'^"= ANYWHERE => PQD(ANYWHERE, OMITABLE,
LOS, "^CP'^")) &\n"^
              genProcess(s,G,P)^
              ")) \n\n" *)
              "(LOS != ANYWHERE & "^L'^"= ANYWHERE => PQD(ANYWHERE, OMITABLE,
LOS, "^CP'^")) &\n"^
              "REACH("^rc^") \n"^
                  ")) &\n"^
              "(REACH("^rc^") => "^(genProcess(s,G,P))^")"
          end
      | genProcess(s,G,Sendp(ML))                =
          let val k = List.length ML
              val Ms = List.map toStringCapability ML
          in
              "/* Sendp */\n"^
```

```
            (String.concat (List.map (fn M => (genM(G,M)^" &\n")) ML))^
            "(A mu. I(mu, "^s^") => \n"^
            "("^quantify(1,k,"A","t")^
               queryRelList(1,"M","t",Ms)^
               " => "^
               "C"^Int.toString(k)^"(mu, "^
               queryVar(1,k,"t")^", EPSI, ANYWHERE)))\n\n"
         end
    | genProcess(s,G,Receivep(ML,XL,P))           =
         let val k = List.length ML
            val l = (List.length XL) + k
            val Ms = List.map toStringCapability ML
            val Ss = List.map toStringNaming XL
            val rc =  newReachCount()
         in
            "/* Receivep */\n"^
            (String.concat (List.map (fn M => (genM(G,M)^" &\n")) ML))^
            "(A mu. I(mu, "^s^") => \n"^
            "("^quantify(1,l,"A","t")^"A ELLO. A LOS. \n"^
               "C"^Int.toString(l)^"(mu, "^queryVar(1,l,"t")^", ELLO, LOS) => "^
               "("^queryRelList(1,"M","t",Ms)^" => "^
               queryRelList(k+1,"R","t",Ss)^" & \n"^
               (* "(LOS != ANYWHERE => PQD(ANYWHERE, OMITABLE, LOS, EPSI)) &\n"^
               genProcess(s,G,P)^
               "))) \n\n" *)
               (*"(LOS != ANYWHERE => PQD(ANYWHERE, OMITABLE, LOS, EPSI)) &\n"^
               genProcess(s,G,P)^
               "))) \n\n" *)
               "(LOS != ANYWHERE => PQD(ANYWHERE, OMITABLE, LOS, EPSI)) &\n"^
               "REACH("^rc^") \n"^
                     ")) &\n"^
               "(REACH("^rc^") => "^(genProcess(s,G,P))^")"
         end
    | genProcess(s,G,Attacker)                 = ""

in

   fun toString (P) = toStringProcess(P)

   fun generate P filename =
         let
                                 val fi = "./test/BAclause."^filename^".cl"
                                 val fh = TextIO.openOut(fi)
                                 val cl = genProcess("TOP",[],P)^" & \n\n"^
                                        "/*   Pending Queries to FI   */\n"^
                  "D(ANYWHERE, OMITABLE) & \n"^
                  "(A LOS. A ELL. A LS. A ELLO. PQD(LS, ELLO, LOS, ELL) => "^
                  "((!D(LOS, ELL) | !D(LS, ELLO)) => FI(ELLO, ELL)))"
                                in
                                  TextIO.output(fh, cl);
                                  TextIO.closeOut(fh)
                                end
end
end
```

Appendix F Source Code of the Processes of Protocols

F.1 WMF

```
CM.make' "sources.cm";

structure MLWMF =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kas  = Name("kas_",[i])
   val gKas = ("keyAS_",[i])

   val kbs  = Name("kbs_",[j])
   val gKbs = ("keyBS_",[j])

   (* Sites *)
   val a        = Name("a_",[i])
   val gAlice   = ("alice",[i])

   val b        = Name("b",[j])
   val gBob     = ("bob",[j])

   val s        = Name("s",[])
   val gServer  = ("server",[])

   (* Message *)
   val m1       = Name("m1",[i,j])
   val gM1      = ("msg",[i,j])

   (* Variables *)
   val x    = Var("x",[i,j])
   val ykab = Var("ykab",[i,j])
   val zkab = Var("zkab",[i,j])

   (* Cryption-point *)
   val a1 = ("A1",[i,j])
   val a2 = ("A2",[i,j])
   val b1 = ("B1",[i,j])
   val b2 = ("B2",[i,j])
   val s1 = ("S1",[i,j])
   val s2 = ("S2",[i,j])

   (* local communication label *)
   val la1 = ("LA1",[i,j])
   val la2 = ("LA2",[i,j])
   val lb1 = ("LB1",[i,j])
   val lb2 = ("LB2",[i,j])
   val ls1 = ("LS1",[i,j])
   val ls2 = ("LS2",[i,j])
```

```
(* Sites *)
val A = IPar("i",NATURAL3,
        Amb(a,
          IPar("j",NATURAL3,
            New(kab,gKab,
              Par(Amb(kas,Pre(Out(a),
                        Pre(In(s),
                          Par(Sendp([NX(a)]),
                            Sendl([NX(b),NX(kab)],a1,[s1],la1)
                          )
                        )
                      )
                  ),
                  New(m1,gM1,
                    Amb(kab,Pre(Out(a),
                          Pre(In(b),
                            Sendl([NX(m1)],a2,[b2],la2)
                          )
                        )
                    )
                  )
              )  (* Par *)
            )  (* New kab *)
          )
        ) (* Ambient a *)
      )  (* IPar *)

val B = IPar("j",NATURAL3,
        Amb(b,
          IPar("i",NATURAL3,
            Receivec([NX(a)],[zkab],kbs,b1,[s2],lb1,
                 Receivec([],[x],zkab,b2,[a2],lb2,Nil)
            )
          )
        )
      )

val S = Amb(s,
        IPar("i",NATURAL3,
          IPar("j",NATURAL3,
            Receivel([NX(a)],[],
              Receivec([NX(b)],[ykab],kas,s1,[a1],ls1,
                Amb(kbs,
                  Pre(Out(s),
                    Pre(In(b),
                      Sendl([NX(a),NX(ykab)],s2,[b1],ls2)
                    )
                  )
                )
              )
            )
          )
        )
      )

val P = INews([("i",NATURAL3)],a,gAlice,
          INews([("j",NATURAL3)],b,gBob,
            News(s,gServer,
              Par(
                INew([("i",NATURAL3)],kas,gKas,
                  INew([("j",NATURAL3)],kbs,gKbs,
                    Par(A,
                      Par(B,S)
                    )
                  )
                ),
                Attacker
              )
```

```
                )
            )
        )
in
   fun analyse () = ANALYSIS.generate P "mlwmf"
   fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

WMF Variant 1:

```
CM.make' "sources.cm";

structure MLWMF1 =

struct
local
    open MLBA
    open ANALYSIS

    (* Namings and their groups *)
    val i = "i"
    val j = "j"

    (* private keys *)
    val kab  = Name("kab_",[i,j])
    val gKab = ("keyAB_",[i,j])

    val kas  = Name("kas_",[i])
    val gKas = ("keyAS_",[i])

    val kbs  = Name("kbs_",[j])
    val gKbs = ("keyBS_",[j])

    (* Sites *)
    val a        = Name("a_",[i])
    val gAlice   = ("alice",[i])

    val b        = Name("b",[j])
    val gBob     = ("bob",[j])

    val s        = Name("s",[])
    val gServer  = ("server",[])

    (* Message *)
    val m1       = Name("m1",[i,j])
    val gM1      = ("msg",[i,j])

    (* Variables *)
    val x    = Var("x",[i,j])
    val ykab = Var("ykab",[i,j])
    val zkab = Var("zkab",[i,j])

    (* Cryption-point *)
    val a1 = ("A1",[i,j])
    val a2 = ("A2",[i,j])
    val b1 = ("B1",[i,j])
    val b2 = ("B2",[i,j])
    val s1 = ("S1",[i,j])
    val s2 = ("S2",[i,j])

    (* local communication label *)
    val la1 = ("LA1",[i,j])
    val la2 = ("LA2",[i,j])
    val lb1 = ("LB1",[i,j])
    val lb2 = ("LB2",[i,j])
    val ls1 = ("LS1",[i,j])
    val ls2 = ("LS2",[i,j])
```

```
(* Sites *)
val A = IPar("i",NATURAL3,
          Amb(a,
             IPar("j",NATURAL3,
               New(kab,gKab,
                  Par(Amb(kas,Pre(Out(a),
                                 Pre(In(s),
                                    Par(Sendp([NX(a)]),
                                       Sendl([NX(b),NX(kab)],a1,[s1],la1)
                                    )
                                 )
                              )
                         ),
                         New(m1,gM1,
                            Amb(kab,Pre(Out(a),
                                       Pre(In(b),
                                          Sendl([NX(m1)],a2,[b2],la2)
                                       )
                                    )
                            )
                         )
                  )  (* Par *)
               )  (* New kab *)
             )
          ) (* Ambient a *)
        )  (* IPar *)

val B = IPar("j",NATURAL3,
          Amb(b,
            IPar("i",NATURAL3,
               Receivel([NX(a)],[],
                 Receivec([],[zkab],kbs,b1,[s2],lb1,
                         Receivec([],[x],zkab,b2,[a2],lb2,Nil)
                 )
               )
            )
          )
        )

val S = Amb(s,
          IPar("i",NATURAL3,
            IPar("j",NATURAL3,
              Receivel([NX(a)],[],
                Receivec([NX(b)],[ykab],kas,s1,[a1],ls1,
                  Amb(kbs,
                     Pre(Out(s),
                        Pre(In(b),
                           Par(Sendp([NX(a)]),Sendl([NX(ykab)],s2,[b1],ls2))
                        )
                     )
                  )
                )
              )
            )
          )
        )

val P = INews([("i",NATURAL3)],a,gAlice,
            INews([("j",NATURAL3)],b,gBob,
                 News(s,gServer,
                     Par(
                        INew([("i",NATURAL3)],kas,gKas,
                            INew([("j",NATURAL3)],kbs,gKbs,
                              Par(A,
                                 Par(B,S)
                              )
                            )
                        ),
                    Attacker
                  )
               )
            )
         )
in
   fun analyse () = ANALYSIS.generate P "mlwmf_v1"
   fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

WMF Variant 2:

```
CM.make' "sources.cm";

structure MLWMF2 =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kas  = Name("kas_",[i])
   val gKas = ("keyAS_",[i])

   val kbs  = Name("kbs_",[j])
   val gKbs = ("keyBS_",[j])

   (* Sites *)
   val a        = Name("a_",[i])
   val gAlice   = ("alice",[i])

   val b        = Name("b",[j])
   val gBob     = ("bob",[j])

   val s        = Name("s",[])
   val gServer  = ("server",[])

   (* Message *)
   val m1       = Name("m1",[i,j])
   val gM1      = ("msg",[i,j])

   (* Variables *)
   val x    = Var("x",[i,j])
   val ykab = Var("ykab",[i,j])
   val zkab = Var("zkab",[i,j])

   (* Cryption-point *)
   val a1 = ("A1",[i,j])
   val a2 = ("A2",[i,j])
   val b1 = ("B1",[i,j])
   val b2 = ("B2",[i,j])
   val s1 = ("S1",[i,j])
   val s2 = ("S2",[i,j])

   (* local communication label *)
   val la1 = ("LA1",[i,j])
   val la2 = ("LA2",[i,j])
   val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val ls1 = ("LS1",[i,j])
val ls2 = ("LS2",[i,j])

(* Sites *)
val A = IPar("i",NATURAL3,
        Amb(a,
           IPar("j",NATURAL3,
             New(kab,gKab,
                Par(Amb(kas,Pre(Out(a),
                               Pre(In(s),
                                  Par(Sendp([NX(a),NX(b)]),
                                     Sendl([NX(kab)],a1,[s1],la1)
                                  )
                               )
                            )
                       ),
                       New(m1,gM1,
                          Amb(kab,Pre(Out(a),
                                     Pre(In(b),
                                        Sendl([NX(m1)],a2,[b2],la2)
                                     )
                                  )
                          )
                       )
                )  (* Par *)
             )  (* New kab *)
           )
        ) (* Ambient a *)
      )  (* IPar *)

val B = IPar("j",NATURAL3,
        Amb(b,
          IPar("i",NATURAL3,
             Receivec([NX(a)],[zkab],kbs,b1,[s2],lb1,
                   Receivec([],[x],zkab,b2,[a2],lb2,Nil)
             )
          )
        )
      )

val S = Amb(s,
        IPar("i",NATURAL3,
          IPar("j",NATURAL3,
            Receivel([NX(a),NX(b)],[],
              Receivec([],[ykab],kas,s1,[a1],ls1,
                Amb(kbs,
                   Pre(Out(s),
                      Pre(In(b),
                         Sendl([NX(a),NX(ykab)],s2,[b1],ls2)
                      )
                   )
                )
              )
            )
          )
        )
      )

val P = INews([("i",NATURAL3)],a,gAlice,
          INews([("j",NATURAL3)],b,gBob,
               News(s,gServer,
                  Par(
                     INew([("i",NATURAL3)],kas,gKas,
                        INew([("j",NATURAL3)],kbs,gKbs,
                          Par(A,
                             Par(B,S)
                          )
                        )
```

```
                    ),
                    Attacker
                  )
                )
              )
           )
in
   fun analyse () = ANALYSIS.generate P "mlwmf_v2"
   fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

F.2 Needham Schroeder

```
CM.make' "sources.cm";

structure MLNHS =

struct
local
    open MLBA
    open ANALYSIS

    (* Namings and their groups *)
    val i = "i"
    val j = "j"

    (* private keys *)
    val kab  = Name("kab_",[i,j])
    val gKab = ("keyAB_",[i,j])

    val kas  = Name("kas_",[i])
    val gKas = ("keyAS_",[i])

    val kbs  = Name("kbs_",[j])
    val gKbs = ("keyBS_",[j])

    (* Nounce or random number *)
    val ra   = Name("ra_",[i,j])
    val gRa  = ("randomA_",[i,j])

    val rb   = Name("rb_",[i,j])
    val gRb  = ("randomB_",[i,j])

    (* anonymous packet *)
    val p    = Name("p_",[])
    val gP   = ("packet_",[])

    (* Sites *)
    val a        = Name("a_",[i])
    val gAlice   = ("alice",[i])

    val b        = Name("b",[j])
    val gBob     = ("bob",[j])

    val s        = Name("s",[])
    val gServer  = ("server",[])

    (* Message *)
    val m1       = Name("m1",[i,j])
    val gM1      = ("msg",[i,j])

    (* Variables *)
    val xkab = Var("xkab",[i,j])
    val xrb  = Var("xrb",[i,j])
    val yra  = Var("yra",[i,j])
val zkab = Var("zkab",[i,j])
val zm   = Var("zm",[i,j])

(* Cryption-point *)
val a1 = ("A1",[i,j])
val a2 = ("A2",[i,j])
val a3 = ("A3",[i,j])
val a4 = ("A4",[i,j])
val b1 = ("B1",[i,j])
val b2 = ("B2",[i,j])
val b3 = ("B3",[i,j])
val b4 = ("B4",[i,j])
val s1 = ("S1",[i,j])
val s2 = ("S2",[i,j])

(* local communication label *)
val la1 = ("LA1",[i,j])
val la2 = ("LA2",[i,j])
val la3 = ("LA3",[i,j])
val la4 = ("LA4",[i,j])
val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val lb3 = ("LB3",[i,j])
val lb4 = ("LB4",[i,j])
val ls1 = ("LS1",[i,j])
val ls2 = ("LS2",[i,j])

(* Sites *)
val A = IPar("i",NATURAL3,
         Amb(a,
           IPar("j",NATURAL3,
            New(ra,gRa,
              Par(Amb(p,Pre(Out(a),
                         Pre(In(s),
                           Sendp([NX(a),NX(b),NX(ra)])
                         )
                       )
                  ),
                  Par(Receivec([NX(a)],[xkab],kas,a1,[s1],la1,
                        Receivec([],[xrb],xkab,a2,[b2],la2,
                          Amb(xkab,Pre(Out(a),
                                  Pre(In(b),
                                    Sendl([NX(xrb),NX(xrb)],a3,[b3],la3)
                                  )
                                )
                          )
                        )
                      ),
                      New(m1,gM1,
                        Amb(xkab,Pre(Out(a),
                               Pre(In(b),
                                 Sendl([NX(m1)],a4,[b4],la4)
                               )
                             )
                        )
                      )
                  )  (* Par *)
              )  (* Par *)
            )  (* New ra *)
           )
         ) (* Ambient a *)
       )  (* IPar *)

val B = IPar("j",NATURAL3,
         Amb(b,
          IPar("i",NATURAL3,
            Receivec([NX(a)],[zkab],kbs,b1,[s2],lb1,
             New(rb,gRb,
               Par(
                        Amb(zkab,Pre(Out(b),
                                  Pre(In(a),
                                     Sendl([NX(rb)],b2,[a2],lb2)
                                  )
                                )
                        ),
                        Receivec([NX(rb),NX(rb)],[],zkab,b3,[a3],lb3,
                          Receivec([],[zm],zkab,b4,[a4],lb4,Nil)
                        )
                      )  (* Par *)
                   )  (* New *)
                 )  (* Receivec *)
               )
             )
           )

   val S = Amb(s,
             IPar("i",NATURAL3,
               IPar("j",NATURAL3,
                 Receivel([NX(a),NX(b)],[yra],
                   New(kab,gKab,
                     Amb(kas,
                        Pre(Out(s),
                           Pre(In(a),
                             Par(
                               Sendl([NX(yra),NX(kab)],s1,[a1],ls1),
                               Amb(kbs,
                                  Pre(Out(kas),
                                     Pre(Out(a),
                                        Pre(In(b),
                                           Sendl([NX(yra),NX(kab)],s1,[a1],ls1)
                                        )
                                     )
                                  )
                               )  (* Amb *)
                             )  (* Par *)
                           )
                        )
                     )  (* Amb *)
                   )  (* New *)
                 )  (* Receivec *)
               )
             )
           )

   val P = INews([("i",NATURAL3)],a,gAlice,
             INews([("j",NATURAL3)],b,gBob,
                 News(s,gServer,
                    Par(
                       New(p,gP,
                         INew([("i",NATURAL3)],kas,gKas,
                            INew([("j",NATURAL3)],kbs,gKbs,
                              Par(A,
                                 Par(B,S)
                              )
                            )
                         )
                       ),
                       Attacker
                    )
                 )
             )
           )
in
   fun analyse () = ANALYSIS.generate P "mlnhs"
   fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

Correcting the flaw

```
CM.make' "sources.cm";

structure MLNHS_COR =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   (* Index *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kas  = Name("kas_",[i])
   val gKas = ("keyAS_",[i])

   val kbs  = Name("kbs_",[j])
   val gKbs = ("keyBS_",[j])

   (* Nounce or random number *)
   val ra   = Name("ra_",[i,j])
   val gRa  = ("randomA_",[i,j])

   val rb   = Name("rb_",[i,j])
   val gRb  = ("randomB_",[i,j])

   (* padding message to correct the flaw *)
   val u1   = Name("u1_",[i,j])
   val gU1  = ("u1_",[i,j])
   val u2   = Name("u2_",[i,j])
   val gU2  = ("u2_",[i,j])

   (* anonymous packet *)
   val p    = Name("p_",[])
   val gP   = ("packet_",[])

   (* Sites *)
   val a        = Name("a_",[i])
   val gAlice   = ("alice",[i])

   val b        = Name("b",[j])
   val gBob     = ("bob",[j])

   val s        = Name("s",[])
   val gServer  = ("server",[])

   (* Message *)
   val m1       = Name("m1",[i,j])
   val gM1      = ("msg",[i,j])

   (* Variables *)
   val xkab = Var("xkab",[i,j])
   val xrb  = Var("xrb",[i,j])
   val yra  = Var("yra",[i,j])
   val zkab = Var("zkab",[i,j])
   val zm   = Var("zm",[i,j])

   (* Cryption-point *)
   val a1 = ("A1",[i,j])
   val a2 = ("A2",[i,j])
   val a3 = ("A3",[i,j])
val a4 = ("A4",[i,j])
val b1 = ("B1",[i,j])
val b2 = ("B2",[i,j])
val b3 = ("B3",[i,j])
val b4 = ("B4",[i,j])
val s1 = ("S1",[i,j])
val s2 = ("S2",[i,j])

(* local communication label *)
val la1 = ("LA1",[i,j])
val la2 = ("LA2",[i,j])
val la3 = ("LA3",[i,j])
val la4 = ("LA4",[i,j])
val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val lb3 = ("LB3",[i,j])
val lb4 = ("LB4",[i,j])
val ls1 = ("LS1",[i,j])
val ls2 = ("LS2",[i,j])

(* Sites *)
val A = IPar("i",NATURAL3,
          Amb(a,
            IPar("j",NATURAL3,
              New(ra,gRa,
                Par(Amb(p,Pre(Out(a),
                          Pre(In(s),
                            Sendp([NX(a),NX(b),NX(ra)])
                          )
                        )
                   ),
                   Par(Receivec([NX(a)],[xkab],kas,a1,[s1],la1,
                         Receivec([NX(u1)],[xrb],xkab,a2,[b2],la2,
                           Amb(xkab,Pre(Out(a),
                                  Pre(In(b),
                                    Sendl([NX(u2),NX(xrb),NX(xrb)],a3,[b3],la3)
                                  )
                                )
                           )
                         )
                      ),
                      New(m1,gM1,
                        Amb(xkab,Pre(Out(a),
                              Pre(In(b),
                                Sendl([NX(m1)],a4,[b4],la4)
                              )
                            )
                        )
                      )
                   )  (* Par *)
                )  (* Par *)
              )  (* New ra *)
            )
          ) (* Ambient a *)
        )  (* IPar *)

val B = IPar("j",NATURAL3,
          Amb(b,
           IPar("i",NATURAL3,
             Receivec([NX(a)],[zkab],kbs,b1,[s2],lb1,
              New(rb,gRb,
                Par(
                  Amb(zkab,Pre(Out(b),
                          Pre(In(a),
                            Sendl([NX(u1),NX(rb)],b2,[a2],lb2)
                          )
                        )
                  ),
                  Receivec([NX(u2),NX(rb),NX(rb)],[],zkab,b3,[a3],lb3,
                  Receivec([],[zm],zkab,b4,[a4],lb4,Nil)
                 )
                ) (* Par *)
               ) (* New *)
              ) (* Receivec *)
             )
            )
           )

  val S = Amb(s,
          IPar("i",NATURAL3,
           IPar("j",NATURAL3,
            Receivel([NX(a),NX(b)],[yra],
             New(kab,gKab,
              Amb(kas,
                Pre(Out(s),
                  Pre(In(a),
                   Par(
                    Sendl([NX(yra),NX(kab)],s1,[a1],ls1),
                    Amb(kbs,
                      Pre(Out(kas),
                        Pre(Out(a),
                          Pre(In(b),
                            Sendl([NX(yra),NX(kab)],s2,[b1],ls2)
                          )
                        )
                      )
                    ) (* Amb *)
                   ) (* Par *)
                  )
                 )
               ) (* Amb *)
              ) (* New *)
             ) (* Receivec *)
            )
           )
          )

  val P = INews([("i",NATURAL3)],a,gAlice,
           INews([("j",NATURAL3)],b,gBob,
            INew([("i",NATURAL3),("j",NATURAL3)],u1,gU1,
            INew([("i",NATURAL3),("j",NATURAL3)],u2,gU2,
             News(s,gServer,
               Par(
                 New(p,gP,
                  INew([("i",NATURAL3)],kas,gKas,
                    INew([("j",NATURAL3)],kbs,gKbs,
                     Par(A,
                       Par(B,S)
                     )
                    )
                   )
                 ),
                 Attacker
               )
              )
             ))
            )
           )
in
  fun analyse () = ANALYSIS.generate P "mlnhs_cor"
  fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

F.3 Otway-Rees.

```
CM.make' "sources.cm";

structure MLOTW =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   (* Index *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kas  = Name("kas_",[i])
   val gKas = ("keyAS_",[i])

   val kbs  = Name("kbs_",[j])
   val gKbs = ("keyBS_",[j])

   (* Nounce or random number *)
   val n    = Name("n_",[i,j])
   val gN   = ("nounceN_",[i,j])

   val na   = Name("na_",[i,j])
   val gNa  = ("nounceNa_",[i,j])

   val nb   = Name("nb_",[i,j])
   val gNb  = ("nounceNb_",[i,j])

   (* Sites *)
   val a        = Name("a_",[i])
   val gAlice   = ("alice",[i])

   val b        = Name("b",[j])
   val gBob     = ("bob",[j])

   val s        = Name("s",[])
   val gServer  = ("server",[])

   (* Message *)
   val m1       = Name("m1",[i,j])
   val gM1      = ("msg",[i,j])

   (* Variables *)
   val xkab = Var("xkab",[i,j])
   val yn   = Var("yn",[i,j])
   val yna  = Var("yna",[i,j])
   val ynb  = Var("ynb",[i,j])
   val zn   = Var("zn",[i,j])
   val zkab = Var("zkab",[i,j])
   val zm   = Var("zm",[i,j])

   (* Cryption-point *)
   val a1 = ("A1",[i,j])
   val a2 = ("A2",[i,j])
   val a3 = ("A3",[i,j])
   val a4 = ("A4",[i,j])
   val b1 = ("B1",[i,j])
   val b2 = ("B2",[i,j])
   val b3 = ("B3",[i,j])
   val b4 = ("B4",[i,j])
val s1 = ("S1",[i,j])
val s2 = ("S2",[i,j])
val s3 = ("S3",[i,j])
val s4 = ("S4",[i,j])

(* local communication label *)
val la1 = ("LA1",[i,j])
val la2 = ("LA2",[i,j])
val la3 = ("LA3",[i,j])
val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val lb3 = ("LB3",[i,j])
val ls1 = ("LS1",[i,j])
val ls2 = ("LS2",[i,j])
val ls3 = ("LS3",[i,j])
val ls4 = ("LS4",[i,j])

(* Sites *)
val A = IPar("i",NATURAL3,
          Amb(a,
            IPar("j",NATURAL3,
             New(n,gN,New(na,gNa,
               Par(Amb(kas,Pre(Out(a),
                          Pre(In(b),
                            Par(Sendp([NX(n)]),
                              Pre(Out(b),
                                Pre(In(s),
                                  Par(Sendp([NX(n)]),
                                    Sendl([NX(a),NX(b),NX(n),NX(na)],a1,[s1],la1)
                                  )
                                )
                              )
                            )
                          )
                        )
                   ),
                   Receivel([NX(n)],[],
                        Receivec([NX(na)],[xkab],kas,a2,[s4],la2,
                          New(m1,gM1,
                            Amb(xkab,Pre(Out(a),
                                   Pre(In(b),
                                     Sendl([NX(m1)],a3,[b3],la3)
                                   )
                                  )
                            )
                          )
                        )
                   )
               )  (* Par *)
             ))  (* New n and na *)
            )
          ) (* Ambient a *)
        )  (* IPar *)

val B = IPar("j",NATURAL3,
          Amb(b,
           IPar("i",NATURAL3,
             Receivel([],[zn],
              New(nb,gNb,
                Par(
                   Amb(kbs,Pre(Out(b),
                          Pre(In(s),
                            Sendl([NX(a),NX(b),NX(zn),NX(nb)],b1,[s2],lb1)
                          )
                        )
                   ),
                   Receivel([],[zn],
                     Receivec([NX(nb)],[zkab],kbs,b2,[s3],lb2,
                       Receivec([],[zm],zkab,b3,[a3],lb3,Nil)
```

```
                    )
                  )
                )  (* Par *)
              )  (* New *)
            )  (* Receivel *)
          )
        )
      )

  val S = Amb(s,
          IPar("i",NATURAL3,
            IPar("j",NATURAL3,
              Receivel([],[yn],
                Receivec([NX(a),NX(b),NX(yn)],[yna],kas,s1,[a1],ls1,
                  Receivec([NX(a),NX(b),NX(yn)],[ynb],kbs,s2,[b1],ls2,
                    New(kab,gKab,
                      Amb(kbs,
                        Pre(Out(s),
                          Pre(In(b),
                            Par(
                              Sendp([NX(yn)]),
                              Par(
                                Sendl([NX(ynb),NX(kab)],s3,[b2],ls3),
                                  Amb(kas,
                                    Pre(Out(kbs),
                                      Pre(Out(b),
                                        Pre(In(a),
                                          Par(
                                            Sendp([NX(yn)]),
                                            Sendl([NX(yna),NX(kab)],s4,[a2],ls4)
                                          )
                                        )
                                      )
                                    )
                                  )  (* Amb *)
                                )  (* Sendl *)
                              )  (* Par *)
                            )  (* Par *)
                          )
                        )
                      )  (* Amb *)
                    )  (* New *)
                  )
                )
              )  (* Receivel *)
            )
          )
        )

  val P = INews([("i",NATURAL3)],a,gAlice,
            INews([("j",NATURAL3)],b,gBob,
              News(s,gServer,
                Par(
                  INew([("i",NATURAL3)],kas,gKas,
                    INew([("j",NATURAL3)],kbs,gKbs,
                      Par(A,
                        Par(B,S)
                      )
                    )
                  ),
                  Attacker
                )
              )
            )
          )
in
  fun analyse () = ANALYSIS.generate P "mlotw"
  fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

F.4 Yahalom

```
CM.make' "sources.cm";

structure MLYAH =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   (* Index *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kas  = Name("kas_",[i])
   val gKas = ("keyAS_",[i])

   val kbs  = Name("kbs_",[j])
   val gKbs = ("keyBS_",[j])

   (* anonymous packet *)
   val p    = Name("p_",[])
   val gP   = ("packet_",[])

   (* Nounce or random number *)
   val na   = Name("na_",[i,j])
   val gNa  = ("nounceNa_",[i,j])

   val nb   = Name("nb_",[i,j])
   val gNb  = ("nounceNb_",[i,j])

   (* Sites *)
   val a       = Name("a_",[i])
   val gAlice   = ("alice",[i])

   val b       = Name("b",[j])
   val gBob     = ("bob",[j])

   val s       = Name("s",[])
   val gServer  = ("server",[])

   (* Message *)
   val m1      = Name("m1",[i,j])
   val gM1      = ("msg",[i,j])

   (* Variables *)
   val xkab = Var("xkab",[i,j])
   val xnb  = Var("xnb",[i,j])
   val yn   = Var("yn",[i,j])
   val yna  = Var("yna",[i,j])
   val ynb  = Var("ynb",[i,j])
   val zna  = Var("zna",[i,j])
   val zkab = Var("zkab",[i,j])
   val zm   = Var("zm",[i,j])

   (* Cryption-point *)
   val a1 = ("A1",[i,j])
   val a2 = ("A2",[i,j])
   val a3 = ("A3",[i,j])
   val b1 = ("B1",[i,j])
val b2 = ("B2",[i,j])
val b3 = ("B3",[i,j])
val b4 = ("B4",[i,j])
val s1 = ("S1",[i,j])
val s2 = ("S2",[i,j])
val s3 = ("S3",[i,j])

(* local communication label *)
val la1 = ("LA1",[i,j])
val la2 = ("LA2",[i,j])
val la3 = ("LA3",[i,j])
val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val lb3 = ("LB3",[i,j])
val lb4 = ("LB4",[i,j])
val ls1 = ("LS1",[i,j])
val ls2 = ("LS2",[i,j])
val ls3 = ("LS3",[i,j])

(* Sites *)
val A = IPar("i",NATURAL1,
        Amb(a,
          IPar("j",NATURAL1,
            New(na,gNa,
              Par(Amb(p,Pre(Out(a),
                          Pre(In(s),
                            Sendp([NX(a),NX(na)])
                          )
                        )
                  ),
                  Par(
                    Receivec([NX(b),NX(na)],[xkab,xnb],kas,a1,[s2],la1,
                      Amb(xkab,Pre(Out(a),
                                Pre(In(b),
                                  Sendl([NX(xnb)],a2,[b3],la2)
                                )
                              )
                      )
                    ),
                    New(m1,gM1,
                      Amb(xkab,Pre(Out(a),
                                Pre(In(b),
                                  Sendl([NX(m1)],a3,[b4],la3)
                                )
                              )
                      )
                    )
                  )
              )  (* Par *)
            )  (* New na *)
          )
        ) (* Ambient a *)
      )  (* IPar *)

val B = IPar("j",NATURAL1,
        Amb(b,
          IPar("i",NATURAL1,
            Receivel([NX(a)],[zna],
              New(nb,gNb,
                Par(
                  Amb(kbs,Pre(Out(b),
                          Pre(In(s),
                            Par(Sendp([NX(b)]),
                              Sendl([NX(a),NX(zna),NX(nb)],b1,[s1],lb1)
                            )
                          )
                        )
                  ),
                  Receivec([NX(a)],[zkab],kbs,b2,[s3],lb2,
                    Receivec([NX(nb)],[],zkab,b3,[a2],lb3,
                     Receivec([],[zm],zkab,b4,[a3],lb4,Nil)
                    )
                   )
                  )  (* Par *)
                 )  (* New *)
                )  (* Receivel *)
               )
              )
             )

   val S = Amb(s,
          IPar("i",NATURAL1,
           IPar("j",NATURAL1,
            Receivel([NX(b)],[],
             Receivec([NX(a)],[yna,ynb],kbs,s1,[b1],ls1,
               New(kab,gKab,
                Amb(kas,
                  Pre(Out(s),
                    Pre(In(a),
                     Par(
                      Sendl([NX(b),NX(yna),NX(kab),NX(ynb)],s2,[a1],ls2),
                      Amb(kbs,
                        Pre(Out(kas),
                          Pre(Out(a),
                            Pre(In(b),
                              Sendl([NX(a),NX(kab)],s3,[b2],ls3)
                            )
                          )
                        )
                      )  (* Amb *)
                     )  (* Par *)
                    )
                  )
                )  (* Amb *)
               )  (* New *)
             )
            )  (* Receivel *)
           )
          )
         )

   val P = INews([("i",NATURAL1)],a,gAlice,
           INews([("j",NATURAL1)],b,gBob,
               News(s,gServer,
                  Par(
                    INew([("i",NATURAL1)],kas,gKas,
                       INew([("j",NATURAL1)],kbs,gKbs,
                         New(p,gP,
                          Par(A,
                             Par(B,S)
                          )
                         )
                       )
                    ),
                    Attacker
                  )
               )
            )
         )
in
   fun analyse () = ANALYSIS.generate P "mlyah"
   fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

F.5 Andrew Secure RPC

```
CM.make' "sources.cm";

structure MLANDREW =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   (* Index *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kab' = Name("kab'_",[i,j])
   val gKab'= ("keyAB'_",[i,j])

   (* Nounce or random number *)
   val na   = Name("na_",[i,j])
   val gNa  = ("nounceA_",[i,j])

   val nb   = Name("nb_",[i,j])
   val gNb  = ("nounceB_",[i,j])

   val nb'  = Name("nb'_",[i,j])
   val gNb'  = ("nounceb'_",[i,j])

   (* padding message to correct the flaw *)
   (* val u1   = Name("u1_",[i,j])
   val gU1  = ("u1_",[i,j])
   val u2   = Name("u2_",[i,j])
   val gU2  = ("u2_",[i,j]) *)

   (* anonymous packet *)
   (*val p    = Name("p_",[])
   val gP   = ("packet_",[]) *)

   (* Sites *)
   val a       = Name("a_",[i])
   val gAlice   = ("alice",[i])

   val b       = Name("b",[j])
   val gBob     = ("bob",[j])

   (* Message *)
   val m1      = Name("m1",[i,j])
   val gM1      = ("msg",[i,j])

   (* Variables *)
   val xnb  = Var("xnb",[i,j])
   val xkab'= Var("xkab'",[i,j])
   val xnb'  = Var("xnb'",[i,j])
   val yna  = Var("yna",[i,j])
   val ym   = Var("ym",[i,j])

   (* Cryption-point *)
   val a1 = ("A1",[i,j])
   val a2 = ("A2",[i,j])
   val a3 = ("A3",[i,j])
   val a4 = ("A4",[i,j])
   val a5 = ("A5",[i,j])
   val b1 = ("B1",[i,j])
val b2 = ("B2",[i,j])
val b3 = ("B3",[i,j])
val b4 = ("B4",[i,j])
val b5 = ("B5",[i,j])

(* local communication label *)
val la1 = ("LA1",[i,j])
val la2 = ("LA2",[i,j])
val la3 = ("LA3",[i,j])
val la4 = ("LA4",[i,j])
val la5 = ("LA5",[i,j])
val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val lb3 = ("LB3",[i,j])
val lb4 = ("LB4",[i,j])
val lb5 = ("LB5",[i,j])

(* Sites *)
val A = IPar("i",NATURAL3,
          Amb(a,
            IPar("j",NATURAL3,
             New(na,gNa,
                Par(Amb(kab,Pre(Out(a),
                              Pre(In(b),
                                Par(Sendp([NX(a)]),
                                  Sendl([NX(na)],a1,[b1],la1)
                                )
                              )
                            )
                    ),
                    Par(Receivec([NX(na),NX(na)],[xnb],kab,a2,[b2],la2,
                          Amb(kab,Pre(Out(a),
                                    Pre(In(b),
                                       Sendl([NX(xnb),NX(xnb)],a3,[b3],la3)
                                    )
                                  )
                          )
                        ),
                        New(m1,gM1,
                          Receivec([],[xkab',xnb'],kab,a4,[b4],la4,
                            Amb(xkab',Pre(Out(a),
                                        Pre(In(b),
                                          Sendl([NX(m1)],a5,[b5],la5)
                                        )
                                      )
                            )
                          )
                        )
                    )  (* Par *)
                )  (* Par *)
             )  (* New na *)
            )
          ) (* Ambient a *)
        )  (* IPar *)

val B = IPar("j",NATURAL3,
          Amb(b,
           IPar("i",NATURAL3,
              Receivel([NX(a)],[],
                 Receivec([],[yna],kab,b1,[a1],lb1,
                  New(nb,gNb,
                    Par(
                      Amb(kab,Pre(Out(b),
                                Pre(In(a),
                                   Sendl([NX(yna),NX(yna),NX(nb)],b2,[a2],lb2)
                                )
                              )
                      ),
                      Receivec([NX(nb),NX(nb)],[],kab,b3,[a3],lb3,
                              New(kab',gKab',
                                New(nb',gNb',
                                  Par(
                                    Amb(kab,Pre(Out(b),
                                            Pre(In(a),
                                              Sendl([NX(kab'),NX(nb')],b4,[a4],lb4)
                                            )
                                          )
                                    ),
                                    Receivec([],[ym],kab',b5,[a5],lb5,Nil)
                                  )
                                )
                              )
                            )  (* Receivec *)
                          )  (* Par *)
                        )  (* New *)
                      )  (* Receivec *)
                    )  (* Receivel *)
                  )
                )
              )

  val P = INews([("i",NATURAL3)],a,gAlice,
            INews([("j",NATURAL3)],b,gBob,
              (*INew([("i",NATURAL3),("j",NATURAL3)],u1,gU1,
              INew([("i",NATURAL3),("j",NATURAL3)],u2,gU2,*)
                  Par(
                    INew([("i",NATURAL3),("j",NATURAL3)],kab,gKab,
                      Par(A,B)
                    ),
                    Attacker
                  )
              (*))*)
            )
          )
in
  fun analyse () = ANALYSIS.generate P "mlAndrew"
  fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

Correcting the flaw

```
CM.make' "sources.cm";

structure MLANDREW_COR =

struct
local
   open MLBA
   open ANALYSIS

   (* Namings and their groups *)
   (* Index *)
   val i = "i"
   val j = "j"

   (* private keys *)
   val kab  = Name("kab_",[i,j])
   val gKab = ("keyAB_",[i,j])

   val kab' = Name("kab'_",[i,j])
   val gKab'= ("keyAB'_",[i,j])

   (* Nounce or random number *)
   val na   = Name("na_",[i,j])
   val gNa  = ("nounceA_",[i,j])
val nb   = Name("nb_",[i,j])
val gNb  = ("nounceB_",[i,j])

val nb'   = Name("nb'_",[i,j])
val gNb'  = ("nounceb'_",[i,j])

(* padding message to correct the flaw *)
val u1   = Name("u1_",[i,j])
val gU1  = ("u1_",[i,j])
val u2   = Name("u2_",[i,j])
val gU2  = ("u2_",[i,j])
val u3   = Name("u3_",[i,j])
val gU3  = ("u3_",[i,j])
val u4   = Name("u4_",[i,j])
val gU4  = ("u4_",[i,j])

(* anonymous packet *)
(*val p    = Name("p_",[])
val gP   = ("packet_",[]) *)

(* Sites *)
val a        = Name("a_",[i])
val gAlice   = ("alice",[i])

val b        = Name("b",[j])
val gBob     = ("bob",[j])

(* Message *)
val m1       = Name("m1",[i,j])
val gM1      = ("msg",[i,j])

(* Variables *)
val xnb  = Var("xnb",[i,j])
val xkab'= Var("xkab'",[i,j])
val xnb'  = Var("xnb'",[i,j])
val yna  = Var("yna",[i,j])
val ym   = Var("ym",[i,j])

(* Cryption-point *)
val a1 = ("A1",[i,j])
val a2 = ("A2",[i,j])
val a3 = ("A3",[i,j])
val a4 = ("A4",[i,j])
val a5 = ("A5",[i,j])
val b1 = ("B1",[i,j])
val b2 = ("B2",[i,j])
val b3 = ("B3",[i,j])
val b4 = ("B4",[i,j])
val b5 = ("B5",[i,j])

(* local communication label *)
val la1 = ("LA1",[i,j])
val la2 = ("LA2",[i,j])
val la3 = ("LA3",[i,j])
val la4 = ("LA4",[i,j])
val la5 = ("LA5",[i,j])
val lb1 = ("LB1",[i,j])
val lb2 = ("LB2",[i,j])
val lb3 = ("LB3",[i,j])
val lb4 = ("LB4",[i,j])
val lb5 = ("LB5",[i,j])

(* Sites *)
val A = IPar("i",NATURAL3,
        Amb(a,
          IPar("j",NATURAL3,
            New(na,gNa,
              Par(Amb(kab,Pre(Out(a),
                          Pre(In(b),
                              Par(Sendp([NX(a)]),
                                Sendl([NX(u1),NX(na)],a1,[b1],la1)
                              )
                            )
                          )
                     ),
                     Par(Receivec([NX(u2),NX(na),NX(na)],[xnb],kab,a2,[b2],la2,
                          Amb(kab,Pre(Out(a),
                                 Pre(In(b),
                                   Sendl([NX(u3),NX(xnb),NX(xnb)],a3,[b3],la3)
                                 )
                               )
                          )
                        ),
                        New(m1,gM1,
                          Receivec([NX(u4)],[xkab',xnb'],kab,a4,[b4],la4,
                            Amb(xkab',Pre(Out(a),
                                   Pre(In(b),
                                     Sendl([NX(m1)],a5,[b5],la5)
                                   )
                                 )
                            )
                          )
                        )
                     )  (* Par *)
                   )  (* Par *)
                 )  (* New na *)
               )
             ) (* Ambient a *)
           )  (* IPar *)

   val B = IPar("j",NATURAL3,
             Amb(b,
               IPar("i",NATURAL3,
                  Receivel([NX(a)],[],
                    Receivec([NX(u1)],[yna],kab,b1,[a1],lb1,
                      New(nb,gNb,
                        Par(
                          Amb(kab,Pre(Out(b),
                                 Pre(In(a),
                                   Sendl([NX(u2),NX(yna),NX(yna),NX(nb)],b2,[a2],lb2)
                                 )
                               )
                          ),
                          Receivec([NX(u3),NX(nb),NX(nb)],[],kab,b3,[a3],lb3,
                            New(kab',gKab',
                              New(nb',gNb',
                                Par(
                                  Amb(kab,Pre(Out(b),
                                         Pre(In(a),
                                           Sendl([NX(u4),NX(kab'),NX(nb')],b4,[a4],lb4)
                                         )
                                       )
                                  ),
                                  Receivec([],[ym],kab',b5,[a5],lb5,Nil)
                                )
                              )
                            )
                          )  (* Receivec *)
                        )  (* Par *)
                      )  (* New *)
                    )  (* Receivec *)
                  )  (* Receivel *)
               )
             )
           )

   val P = INews([("i",NATURAL3)],a,gAlice,
             INews([("j",NATURAL3)],b,gBob,
                 Par(
                  INew([("i",NATURAL3),("j",NATURAL3)],u1,gU1,
                  INew([("i",NATURAL3),("j",NATURAL3)],u2,gU2,
                  INew([("i",NATURAL3),("j",NATURAL3)],u3,gU3,
                  INew([("i",NATURAL3),("j",NATURAL3)],u4,gU4,
                    INew([("i",NATURAL3),("j",NATURAL3)],kab,gKab,
                      Par(A,B)
                    )
                  )))),
                  Attacker
                )

            )
        )
in
  fun analyse () = ANALYSIS.generate P "mlAndrew_cor"
  fun analyse1 P filename = ANALYSIS.generate P filename
end
end
```

Appendix G Source Code of Testing

The sml code of the tests of all configurations have been uploaded to the following webpage:

http://www.student.dtu.dk/~s030998/content/research/master_thesis.htm

for your reference.

Appendix G List of Figures

Appendix H List of Tables

Bibliography

[1] Flemming Nielson, Hanne Riis Nielson, Chris Hankin. Principles of Program Analysis. Springer-Verlag, 1999.

[2] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H. Riis Nielson. Static Validation of Security Protocols (to appear), Journal of Computer Security, 2005.

[3] H. Riis Nielson, F. Nielson, M. Buchholtz. *Security for Mobility*, Foundations of Security Analysis and Design II - FOSAD 2001/2002 Tutorial Lectures, vol. 2946, pp. 207-265, 2004.

[4] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols – The Spi calculus. Information and Computation, 148(1):1-70, 1999.

[5] L. Cardelli and A.D. Gordon. Mobile ambients. In Proc. of FoSSaCS'98, LNCS 1378, pages 140-155. Springer, 1998.

[6] L. Cardelli and A. D. Gordon. Mobile Ambients. Theoretical Computer Science, 240(1):177-213, 2000.

[7] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[8] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In Theoretical Aspects in Computer Science (*TACS 2001),* volume 2215 of Lecture Notes in Computer Science, pages 37-63. Springer, 2001.

[9] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. Reasoning about security in Mobile Ambients. *In CONCUR 2001 – Concurrency Theory*, volume 2154 of *Lecture Notes* in *Computer Science*, pages 102-120. Springer, 2001.

[10] F. Nielson, H. Riis Nielson, R.R. Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381-418, 2002.

[11] F. Nielson, H. Riis Nielson, R.R. Hansen, and J.G. Jensen. Validating firewalls in Mobile Ambients. In CONCUR 1999 – *Concurrency* Theory, volume 1664 of *Lecture Notes in Computer Science*, pages 463-477. Springer, 1999.

[12] H. Sun. User's Guide for the Succinct Solver (V2.0). Draft, 2003. http://www2.imm.dtu.dk/cs_SuccinctSolver/userGuide.pdf .

[13] F. Nielson, H. Seidl, and H. Riis Nielson. A Succinct Solver for ALFP, *Nordic journal of computing*, 2002(9), 2002.

[14] H. Riis Nielson and F.Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation*: *Complexity, Analysis, Transformation*, Lecture Notes in Computer Science 2566.pp 223-244. Springer, 2002.

[15] M.S Hecht. *Flow Analysis* of *Computer Programs*. North Holland, 1977.

[16] N. Heintze. Set-based analysis of ML programs. In *Proc. LFP '94*, pages 306-317, 1994.

[17] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In Proc. *POPL '77*, pages 238-252. ACM Press, 1977.

[18] R. Milner. A theory of type polymorphism in programming. *Journal of Computer Systems*, 17:348-375, 1978.

[19] F. Nielson and H. Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proc. POPL '97*, pages 332-345. ACM Press, 1997.

[20] Konstantino Saganos, Terrance Swift, David S. Warren, Juliana Freire, Prasad Rao, Boaqiu Cui, and Ernie Johnson. The *XSB System - Version 2.5 – Programmers Manual*, March 2002.

[21] Konstantinos F. Sagonas, Terrance Swift, and David Scott Warren. Xsb as an e_cient deductive database engine. In Richard T. Snodgrass and Marianne Winslett, *editors, Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994,* pages 442{453. ACM Press, 1994.

[22] H. Pilegaard, *A feasibility Study: The Succinct Solver v2.0, XSB Prolog v2.6, and Flow-Logic Based Program Analysis for Carmel*, 2003.

[23] R.R.Hansen, A Prototype Tool for JavaCard Firewall Analysis, Nordic Workshop on Secure IT-Systems, 2002.

[24] M. Buchholtz. Implementing Control Flow Analysis for security protocols, 2003.

[25] J. Kelsey, B. Schneier, and D. Wagner. Protocol Interactions and the Chosen Protocol Attack, *Security Protocols, 5th International Workshop April 1997 Proceedings*, Springer-Verlag, 1998, pp. 91-104.

[26] F.Levi and S. Maffeis. An abstract interpretation framework for analyzing Mobile Ambients. In *Static Analysis*, 8th *International Symposium*, *SAS 2001*, pages 395-411. Springer, 2001.

[27] D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8-10, 1987.

[28] M. Satyanarayanan. Integrating security in a large distributed system. *ACM ToCS*, 7(3):247-280, 1989.

[29] M. Burrows, M. Abadi, and R. Needham. A logic of Authentication. *ACM Transactions on Computer Systems*, pages 18-36, 1990.

[30] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993-999, 1978.

[31] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols – The Spi calculus. *Information and Computation 148*, 1:1-70, 1999.

[32] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 352-364. ACM Press, 2000.

[33] Hanne Riis Nielson, Flemming Nielson and Henrik Pilegaard. *Spatial Analysis of BioAmbients*, to appear at Static Analysis Symposium (SAS04), Springer Verlag, 2004.

[34] Han Gao. Using the Succinct Solver to Implement Flow Logic Specifications of Classical Data Flow Analyses, *Master's Thesis*, 2004.

[35] David Basin, Sebastian Mödersheim, Luca Viganò. OFMC:A symbolicmodel checker for security protocols, *International Journal of Information Security, pages 181-208*, Springer Verlag, 2005.

[36] F. J. Thayer Fabrega, J. Herzog, and J. D. Guttman, ``Strand spaces: Why is a security protocol correct?," in *1998 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 1998.

[37] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for secrecy and non-interference in networks of processes. In Proc. PaCT'01, number 2127 in Lecture Notes in Computer Science, pages 27-41. Springer-Verlag, 2001.