Static Validation of Voting Protocols

Esben Heltoft Andersen
Christoffer Rosenkilde Nielsen

Kongens Lyngby 2005
IMM-THESIS-2005-55

Summary

Previous studies have shown that language based technologies can be used to automatically validate classical protocols. In this thesis we shall apply these methods to a different type of protocols; namely electronic voting protocols.

We shall study three voting protocols; FOO92, Sensus and E-vox. These are modelled in the process calculus LySA and validated using the corresponding analysis. However, as the protocols utilise cryptographic operations which are not incorporated into LySA, we shall extend the calculus and the analysis. This extension is proven sound with respect to the semantics and the corresponding implementation is proven sound with respect to the analysis.

The difficulties concerning the modelling of the voting protocols in LySA, stresses the need for LySA^{xp}; a process calculus and a corresponding analysis, which we present in the second part of the thesis. The analysis of LySA^{xp} is also proven sound with respect to the semantics.

Resumé

Tidligere studier har vist, at sprogbaserede metoder kan benyttes til at automatisk validere designet af klassiske protokoller. I denne afhandling benytter vi disse metoder til at validere designet af en anderledes type protokoller; elektroniske afstemningsprotokoller.

Vi behandler tre afstemingsprotokoller; FOO92, Sensus og E-Vox. Disse modelleres i proceskalkylen LySA og valideres ved hjælp af den tilhørende analyse. Protokollerne benytter kryptografiske operationer der ikke er inkluderet i LySA, hvorfor kalkyle og analyse må udvides. Udvidelsen af analysen bevises formelt at være sund med hensyn til semantikken og implementeringen bevises sund med hensyn til analysen.

Kompleksiteten i modelleringen af afstemningssystemerne i LySA skaber motiv for LySA^{xp}, en proceskalkyle og analyse der er udviklet som anden del af denne afhandling. Analysen er ligeledes bevist formelt sund med hensyn til semantikken.

Preface

This thesis is part of the work done for obtaining the M.Sc. degree. The work has been carried out at the Department of Informatics and Mathematical Modelling, Technical University of Denmark, under supervision of Professor Hanne Riis Nielson. The project corresponds to 35 ECTS points and ran from February to August, 2005.

Parts of this thesis has been published in the article *Static Validation of a Voting Protocol* [30] which was recently presented in Lisbon, Portugal, at *The Second Workshop on Automated Reasoning for Security Protocol Analysis* (ARSPA'05), one of the satellite workshops of ICALP'05.

Acknowledgements

First of all we would like to thank Hanne Riis Nielson, our supervisor, for encouragement and excellent guidance throughout the project.

We would also like to thank Mikael Buchholtz for tirelessly explaining practically every aspect of LySA; we think we got it now. Our thanks also go to René Rydhof Hansen for his help on developing the analysis for LySA^{XP}, and Flemming Nielson for explaining how to deal with infinities. Furthermore we would like to thank the Language Based Technology group, in particular Henrik Pilegaard and Terkel Kristian Tolstrup for reading and commenting on our

ARSPA'05 article. We would also like to thank the anonymous reviewers from the ARSPA'05 workshop; almost all of their comments have been incorporated in this thesis.

Special thanks to Christoffer's brother; Johan Sebastian Rosenkilde Nielsen, for spending three days reading and commenting the thesis. Finally we are grateful for the support from our families and friends.

Esben Heltoft Andersen Christoffer Rosenkilde Nielsen

Kongens Lyngby, August 2005

Contents

Chapter 1

Introduction

Due to the rapid growth in computer networks, most people nowadays have access to the internet. This makes electronic voting a viable alternative to the classic paper vote for governmental elections as well as small scale elections and surveys. However, the use of electronic voting systems introduces new ways to systematically disrupt the vote or falsify the result. If these systems are to replace the classical way of voting, the communities that hold the elections, should be convinced of their correctness.

The aim of this thesis is to identify what properties are to be fulfilled in an electronic voting system and to provide a framework for validation of these properties.

1.1 Protocol Narrations

In a computer network, as on the Internet, communication proceeds on a public channel known as the *ether*. This communication is handled by *communication protocols*, which describe the rules two or more *principals* in the network follow when they exchange messages. Communication protocols, which rely on cryptographic operations to prevent tampering of the messages sent, are called security protocols.

Protocols are often described using *protocol narrations*, and as an example of this, consider the following protocol narration for a version of the Wide Mouthed Frog protocol (WMF) [3]:

$$\begin{array}{lrcl} 1. & A \rightarrow S & : & A,\ encrypt_{K_A}(B, K) \\ 2. & S \rightarrow B & : & encrypt_{K_B}(A, K) \\ 3. & A \rightarrow B & : & encypt_K(m_1, \cdots, m_k) \end{array}$$

The protocol involves three principals; Alice (A), Bob (B) and a trusted server (S), with whom Alice and Bob each share a secret key; respectively K_A and K_B.

The protocol proceeds in three stages; first Alice sends a message to the server saying that she is A and a message encrypted under the key K_A, saying that she wants to communicate with B using the secret key K. The server responds by sending a message to Bob encrypted under the key K_B, stating that A wants to exchange a message under encryption with the secret key K. In the last step, Alice sends a tuple $(m_1, \cdots, m_k)$ to Bob encrypted under the key K they now share.

1.2 Framework

The main goal in this thesis is to automatically validate electronic voting protocols, and to do this we need some sort of strategy. Our strategy is illustrated with the framework in Figure 1.1.

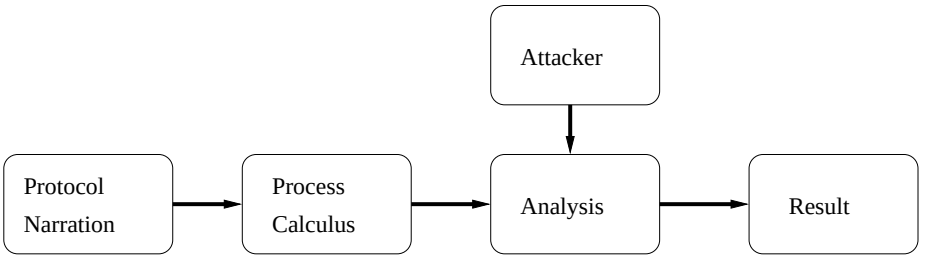


Figure 1.1: Framework

To formally validate a protocol, we need a mathematical model of the protocol instead of the informal protocol narration. In our case we use process calculi to model the protocol, and we use this mathematical model of the protocol to formally validate it. The technique used for this validation is known as program analysis [31].

In addition to a model of the protocol, we shall also model the environment in which the protocol is evaluated. As mentioned above the communication medium we consider, is a public channel; the ether. Our model of the environment must be realistic with respect to the actual behavior that could occur in such a network, and thus malicious behavior of the environment is represented as an *attacker* model.

As already presented in the narration for the WMF protocol, we shall model cryptographic primitives as algebraic terms. This approach is widely used in analyses of security protocols and implies the assumption of *perfect cryptography*; that encrypted terms can only be decrypted using the correct key.

The protocol narration in our framework was described above, and in the following we shall describe the remaining three elements of the framework; process calculi, analysis technique and the attacker.

1.2.1 Process Calculi

Process calculi has been used over several decades to describe the concepts of computational functions such as programs and protocols; Tony Hoare's CSP [22] and Robin Milner's π-calculus [28] are examples of such calculi.

There are two issues one must consider when choosing or developing a calculus; on one hand, the calculus must be expressive in a way, such that one is able to model the precise intention of the program. On the other hand the calculus must be simple in its syntax such that encoding programs can be done easily, and simple in its semantics such that the analysis can be developed without an unnecessary complexity.

A number of calculi for analysis of security protocols have been developed; the applied pi calculus [2] and the Spi-calculus [3] are well-known calculi with corresponding automatic tools ProVerif [6] and Cryptyc [15]. However the LySA-calculus [7, 8] with an automated analyses incorporated in the LySATool [26] is more suitable for our purpose, as we shall see in this thesis.

1.2.2 Analysis Technique

A protocol can be regarded as a small program, and in that respect we shall use analysis techniques originally developed for programs, to analyse protocols.

Program analysis [31] is used to obtain information about the behavior of programs. As this is a static technique, the approach of program analysis is restricted to give approximate answers. This is illustrated in Figure 1.2, where the dashed circle is the approximation of the program behavior, which is illustrated by the solid line; (a) over-approximation captures the entire behavior of the program, (b) under-approximation ensures that the entire approximation is within the possible behavior of the program, and (c) undecidable approximation does not ensure that the program behavior is in the approximation, nor that the approximation is within the possible behavior of the program, and hence we cannot use an undecidable approximation.

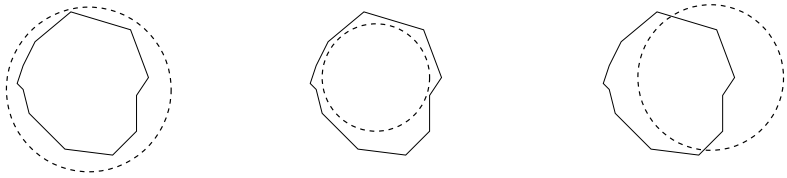


Figure 1.2: (a) Over-approximation (b) Under-approximation (c) Undecidable approximation

The aim of program analysis is to approximate answers and in our case we look for conservative answers; ie. over-approximations. However, an over-approximation possibly also includes behaviors of the program that can never occur in real-life, and if we study one specific behavior in the analysis result, we must be aware that it can be a *false-positive*. Hence a design goal of the analysis is to minimise the number of false-positives, while still maintaining the conservative approximation in the analysis.

That the analysis ensures conservative answers is a result of it being *semantics based*; ie. the information obtained from the analysis can be proven sound with respect to the semantics of the process calculi analysed.

Classical static analysis techniques [31] are normally divided into classes depending on whether they compute the flow of data in a program, the flow of control in a program etc. As process calculi makes it harder to distinguish between flow of control and data in the protocols we model, we refer to our static analysis approach as control flow analysis, even though in reality it is a combination of data flow and control flow analysis.

The specifications of the analyses in this thesis are presented using *Flow Logics*

[34, 35] which are rules of the form:

$$\mathcal{S} \models P \quad \text{iff} \quad \textit{a logic formula holds}$$

This means that the data-structure $\mathcal{S}$, containing the components of the analysis, is an acceptable estimate of the behavior of a process P, if the logical formula to the right of the iff is satisfied.

1.2.3 The Attacker

Previously we stated that malicious behavior in the environment of the protocol can be modelled as an attacker.

Usually cryptography is used to prevent the attacker from learning any of the secrets sent, but if the attacker knows the key, he can of course decrypt messages as well he can create new encryptions of his own. Hence cryptography alone is not enough to protect against attackers, which is why we need a more describing model of the attackers capabilities.

Attackers are divided into two groups; passive and active. A passive attacker is known as an *eavesdropping* attacker. This attacker does not send any message or block any communications on the network, but merely collects information flowing on the network. The active attacker on the other hand, can perform a number of attacks in addition to collecting information; some of these attacks are described in the following.

- *Replay attack* is when an attacker re-sends a message; eg. replaying message 1 in WMF, could get the server to believe that Alice wanted to communicate with Bob again, even though this was not the case. A special kind of replay attack have its own name; namely *Type-flaw attack*. A type-flaw attack is as a replay attack, but where the replayed message is used in another part of the protocol.
- *Modification attack* is when the attacker intercepts a message eg. from A to B and modifies the message before sending the message to B.
- *Man-in-the-Middle attack* is a combinating of a modification attack and a replay attack with the attacker intercepting a message sent from A to B, possibly altering it, before sending it to B thereby making B to believe that the message actually was from A and not from the attacker.
- *Deletion attack* is an attack where the attacker deletes messages from the protocol without the principals noticing it, and would normally lead

to a halting protocol - this attack can usually be handled in a proper implementation of the protocol, and in this thesis it is not considered.

- *Insider attacks* is interesting in the point of view of voting systems. An insider attack is when one of the trusted authorities acts as a malicious attacker on the network.

Obviously the active attacker is superior to the passive, and hence we consider only an active attacker in this thesis.

1.3 Design Goals of Electronic Voting Systems

We will now turn to the specific area of interest in this thesis; electronic voting systems.

There are various approaches in the construction of an electronic voting scheme, but all of them, as well as ordinary paper-ballot elections, are in general divided into the following tasks:

- **Registration** involves creation of a list with eligible voters.
- **Validation** involves checking eligibility of those attempting to vote and validating the ballot for eligible voters.
- **Voting** proceeds on eligible voters sending their ballots which are collected by the authority of the vote.
- **Counting** involves the authority counting the votes in the election.

Voting protocols need reasoning on the properties they have to satisfy. There are properties concerning the environment of the protocol such as convenience and flexibility, and properties concerning the security in the protocol. In this thesis we will focus on the security properties, hence we need to identify what security properties a voting protocol should satisfy. For ordinary protocols, the security properties include authentication, integrity and non-repudiation, but the security properties for electronic voting systems differ from those; in [14] the security properties have been formalised into four main properties:

- **Verifiability**: A system is verifiable if the voters independently can verify that their votes have been counted correctly.

- **Accuracy**: The accuracy of a voting system is divided into three parts: (1) it is not possible for a vote to be altered, (2) a validated vote cannot be eliminated from the final tally and (3) an invalid vote cannot be counted in the final tally.
- **Democracy**: A system ensures democracy if (1) only eligible voters can vote and (2) eligible voters can only vote once.
- **Privacy**: In a voting system the privacy is obtained if nobody can link any vote to the voter who cast it.

Often a fifth property is added [4, 18]:

- **Fairness**: No early results from the vote can be obtained.

The study of proposed electronic voting protocols has revealed that security properties are informally proved and argued to be satisfied.

Because security protocols are notoriously difficult to design and analyse, techniques for formally validating the security properties are particular important. Numerous examples of protocols which were thought to be correct, have been shown to hold major flaws by means of formal validation. The most infamous example of this is the Needham-Schroeder protocol [25]. This motivates the automated analysis of the security properties presented in this thesis.

1.4 Electronic Voting Protocols

Protocols for electronic voting build on different cryptographic primitives, eg. homomorphic encryption [5, 21] and mix-nets [23]. The work presented in this thesis concentrates on protocols using blind signatures [18, 14, 20], but the techniques used for validation could probably be adapted to the other approaches as well.

Blinding [11, 12] is a mechanism allowing a message to be signed by another party, without revealing any information about the message to the other party. Assuming perfect cryptography, blinding is a cryptographic primitive obeying the following two rules:

$$\text{(Unblind 1)} \quad unblind_b(blind_b(msg)) = msg$$
$$\text{(Unblind 2)} \quad unblind_b(sign_s(blind_b(msg))) = sign_s(msg)$$

Here msg is a message, b is a cryptographic key known as the blinding factor and s is a digital signature. The second rule is the most interesting one as it expresses that a signed blinded message can be unblinded without disclosing any information about the message itself; note that the signature of the message is not destroyed. The first rule simply states that blinding acts as symmetric encryption when no signature is present.

One of the first voting protocols using blind signatures was proposed by Fujioka, Okamoto and Ohta [18] in 1992. This protocol is called the FOO92 protocol and is often considered as the classic voting protocol. The FOO92 protocol, as we present below, has served as basis for the two additional voting protocols that we study later in the thesis; Sensus [14] and E-Vox [20] which are presented in Chapter 7.

1.4.1 The FOO92 Voting Protocol

The FOO92 protocol [18] involves three kinds of principals: There are multiple voters V, one administrator A and one counter C. The administrator ensures that only legitimate voters are allowed to vote and the counter collects, publishes and counts the votes. In addition to digital signatures, encryption and blinding, the FOO92 protocol incorporates another cryptographic primitive, bit-commitment [29]. This is a method, by which the voter can commit a vote without revealing what it is. Later the bit can be revealed by the voter, by providing the commitment key. The protocol proceeds in five phases as shown in Table 1.1 and is further explained below.

1.	$V \rightarrow A$	:	$V,\ sign_V(blind_b(commit_r(v)))$	Preparation Phase
2.	$A \rightarrow V$	:	$sign_A(blind_b(commit_r(v)))$	Administration Phase
3.	$(V) \rightarrow C$	:	$sign_A(commit_r(v))$	Voting Phase
4.	$C \rightarrow$	:	$l,\ sign_A(commit_r(v))$	Publishing Phase
5.	$(V) \rightarrow C$	:	$l,\ r$	Opening Phase

Table 1.1: Protocol Narration for FOO92

In the preparation phase (1) the voter V selects vote v and computes the bit-commitment $x = commit_r(v)$ using a random number r and the bit-commitment function $commit$. The commitment is then blinded using the blinding factor b and the resulting $e = blind_b(x)$, called the *ballot*, is signed $s = sign_V(e)$ and sent to the administrator A.

In the administration phase (2) A verifies that V has the right to vote, has not applied for a signature yet and that s actually is V's signature of e. If this is the case then A signs the ballot $d = sign_A(e)$ and sends it back to V.

When V receives the ballot signed by A the voting phase (3) begins. V checks that the signature on the ballot originates from A and unblinds the signed ballot $y = unblind_b(d)$ thereby obtaining a signed version of the committed vote, that is $y = sign_A(x)$. The voter then sends the signed ballot y to the counter over an anonymous communication channel, this is denoted by (V) as the sender in the narration.

In the publishing phase (4) the counter receives y, checks the correctness of the signature and enters (l, y) onto a list as the l-th item. After all votes are received e.g. after a fixed deadline, C publishes the list with all entries.

In the last phase, the opening phase (5), the voter checks that his ballot x is in the list and sends l together with the commitment key r to C on an anonymous channel. When C receives r he is able to open the ballot and count the vote v.

1.5 Overview of the Thesis

The thesis is divided into two parts, each of these parts present an approach to the framework for analysing voting protocols.

Part I. In Chapter 2 we present the LySA-calculus extended with a construct for blinding. We then describe how to model protocols using LySA in Chapter 3, in particular the FOO92 protocol will be specified. The analysis of LySA with blinding is presented in Chapter 4, whereas the attacker-model is presented in Chapter 5. The implementation of the extensions to LySA is described in Chapter 6, and the result of applying the analysis to the three voting protocols is presented in Chapter 7. Finally we discuss the first part of the thesis in Chapter 8

Part II. The second approach to the framework is motivated in Chapter 9. In Chapter 10 the syntax and semantics of LySA$^{\text{XP}}$ is presented, this is followed by Chapter 11 which contains the modelling of the FOO92 protocol in the LySA$^{\text{XP}}$ -calculus. We then present the analysis of LySA$^{\text{XP}}$ in Chapter 12, and the second part of the thesis is discussed in Chapter 13.

Chapter 14 concludes the thesis and discuss the perspectives in the two approaches we have presented.

Part I

LySa

Chapter 2

LySA-Calculus with Blinding

In order to apply our analysis technique, we have to formalise the protocol narration as a process in the LySA-calculus. LySA is a process calculus in the π-calculus tradition [28] and uses ideas from the Spi-calculus [3] for incorporating cryptographic operations. LySA simplifies matters compared to other calculi in that all messages are sent on a global network, the ether, instead of using channels.

In this chapter we present the formal syntax for the LySA-calculus, as well as the semantics for the language, in form of a reduction relation.

2.1 Syntax

The original LySA-calculus was established in [7, 8]. LySA consists of terms and processes, where terms are the basic building blocks used for describing processes. However blinding is a special cryptographic construct, which is not supported by the original calculus, so in order to analyse voting protocols that utilises this primitive, we must extend the syntax for both terms and processes. The resulting syntax for terms is given in Table 2.1 and consists of names, variables and composite terms.

$E ::=$	x	variable
	n	name
	m^+/m^-	public/private keypair
	$\{E_1, \cdots, E_k\}_{E_0}$	symmetric encryption
	$\{\!\{E_1, \cdots, E_k\}\!\}_{E_0}$	asymmetric encryption
	$[\![E_1, \cdots, E_k]\!]_{E_0}$	blinding

Table 2.1: Terms for LySA with blinding

We use x to represent variables. The names n will be used to represent shared keys, commitment keys as well as blinding factors whereas m^+/m^- is used to represent keys for secret key encryptions and digital signatures. As usual digital signatures are obtained using asymmetric encryption with a private key. The special construct $[\![E_1, \cdots, E_k]\!]_{E_0}$ is used for blinding the tuple $E_1, \cdots, E_k$ of terms with the blinding factor E_0.

The syntax for processes, extended such that it respects the blinding construct, is given in Table 2.2.

$P ::=$	0	terminated process
	$P_1 \mid P_2$	parallel composition
	$!P$	replication
	$(\nu\, n)\, P$	restriction (name)
	$(\nu_{\pm}\, m)\, P$	restriction (keypair)
	$\langle E_1, \cdots, E_k \rangle.P$	output
	$(E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k).P$	input
	decrypt E as $\{E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k\}_{E_0}$ in P	symmetric decryption
	decrypt E as $\{\!\{E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k\}\!\}_{E_0}$ in P	asymmetric decryption
	unblind E as $[\![E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k]\!]_{E_0}$ in P	unblinding

Table 2.2: Syntax for LySA with blinding

In addition to the classical constructs for composition, replication and restriction, LySA contains an input construct with matching and the two decryption constructs, and the construct for unblinding follows the same trend.

In the case of input the idea is that the pattern $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k)$ must be matched towards a tuple $(E'_1, \cdots, E'_k)$ of the same length and it only succeeds if the j first components pairwise equals one another, i.e. $E_1 = E'_1, \cdots, E_j = E'_j$. If this is the case the remaining terms $E'_{j+1}, \cdots, E'_k$ are bound to the variables

$x_{j+1}, \cdots, x_k$.

The idea behind the pattern matching of the constructs for symmetric and asymmetric decryption is similar, with the only modification that in the case of symmetric encryption the two keys must be equal whereas for asymmetric encryption they must form a key pair.

Unblinding takes the form unblind E as $[\![E_1, \cdots, E_j; x_{j+1}, \cdots, x_k]\!]_{E_0}$ in P. As already explained in the introduction, the construct may act as an ordinary decryption using the blinding factor as a symmetric key; rule (unblind 1). In this case E must take the form $[\![E'_1, \cdots, E'_k]\!]_{E'_0}$ in order for the construct to succeed, and furthermore the conditions $E_0 = E'_0, E_1 = E'_1, \cdots, E_j = E'_j$ must be fulfilled. When succeeding, the variables $x_{j+1}, \cdots, x_k$ will be bound to $E'_{j+1}, \cdots, E'_k$ as explained above. The more interesting alternative arises when E evaluates to a signed blinded value, i.e. has the form $\{\!|[\![E'_1, \cdots, E'_{k'}]\!]_{E'_0}|\!\}_{E_s}$. In this case the unblinding construct must take the form unblind E as $[\![; x]\!]_{E_0}$ in P and the match will succeed when $E_0 = E'_0$. The variable x will then be bound to the signed value $\{\!|E'_1, \cdots, E'_{k'}|\!\}_{E_s}$ as already illustrated by the rule (Unblind 2) in the introduction.

2.2 Semantics

Following the π-calculus tradition, the semantics of LySA is given as a reduction semantics that describes how a process evolves in a step-by-step fashion. This is formalised in a binary relation called the *reduction relation*. The reduction relation holds between a pair of processes, written $P \rightarrow P'$, precisely when P can evolve into P'.

Reduction relations typically require a process to be on a specific form to match the rules. Therefore, in order to loosen up these rigid requirements, some syntactic manipulations of processes must be introduced, before moving to the reduction relation itself. These syntactic manipulations has been introduced in a form of *structural congruence*, written $P \equiv P'$. The idea of this relation is that two processes are considered equal, when they only differ in syntactic aspects that are of no importance to the way the processes may evolve. Structural congruence is defined as the smallest relation satisfying the rules in Table A.2 in the appendix. Structural congruence utilise a function $\mathsf{fn}(P)$ for finding free names, which is also defined in the appendix in Table A.4.

The definition of structural congruence is somewhat trivial, except for one case. Two processes P_1 and P_2 are considered structural congruent, if they are α-

$$\text{(UBli1)} \quad \mathsf{unblind}\ [V_1,\ldots,V_k]_{V_0}\ \mathsf{as}\ [V_1,\ldots,V_j;\, x_{j+1},\ldots,x_k]_{V_0}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\ldots,x_k \overset{\alpha}{\mapsto} V_k]$$

$$\text{(UBli2)} \quad \mathsf{unblind}\ \{\!|[V_1,\ldots,V_k]_{V_0}|\!\}_{m^-}\ \mathsf{as}\ [;\, x]_{V_0}\ \mathsf{in}\ P \rightarrow P[x \overset{\alpha}{\mapsto} \{\!|V_1,\ldots,V_k|\!\}_{m^-}]$$

Table 2.3: Reduction relation for blinding; $P \rightarrow P'$.

equivalent, written $P_1 \overset{\alpha}{\equiv} P_2$. That two processes are α-equivalent means that they are identical, except that they possibly differ in the naming of *bound names*. The procedure of replacing all occurrences of a bound name in a process with another name is called α-conversion, and does of course result in an α-equivalent process.

The definition of α-equivalence is is given in Table A.3 in the appendix, and it is important to notice that the substitution $P[n_1 \mapsto n_2]$ only substitutes *free occurrences* of n_1 with n_2 in P; that is occurences of n_1 that are not restricted within the process. Also notice that α-equivalence only applies to names, whereas renaming variables does not result in an α-equivalent process. This choice is made because only α-conversion of free names is necessary for the semantics to work satisfactory.

Returning to the reduction relation, a final ingredient is the substitution of variables for values. Values $V \in \mathsf{Val}$ are simply terms without variables, and the reduction relation is defined such that it substitutes a variable x for a value V whenever x becomes bound to V. This substitution is denoted $P[x \overset{\alpha}{\mapsto} V]$ and substitutes the variable x for the value V in a process P. Again this substitution only applies to free occurrences of x, and furthermore the substitution is *capture avoiding* meaning that no names in V will be captured by a restriction of that name. This is ensured by α-converting restricted names when necessary. For example, assume x occurs free in P, then the substitution in $((\nu\, n)\, P)[x \overset{\alpha}{\mapsto} n]$ requires that n is α-converted before the substitution can be made to avoid confusion between the name in the restriction and the name in the substitution.

We can now present the reduction relation; in Table 2.3, we formalise the functionality of blinding in a reduction relation, as already explained intuitively in Chapter 1 and in presentation of LySA. The full reduction relation for LySA with blinding is listed in Table A.1 in the appendix. The rules (UBli1) and (UBli2) models *perfect blinding*, as unblinding can only occur using the right blinding factor V_0. Again as earlier discussed the rule (UBli1) is identical to the rule for symmetric encryption (SDec) in Table A.1 whereas in the reduction

relation (UBli2) the variable x is bound to the signed value $\{\!|V_1, \cdots, V_k|\!\}_{m^-}$.

2.3 Annotations

To describe the intention of protocols, the terms and syntax of the cryptographic primitives are decorated with labels ℓ called *crypto-points* and *assertions* of one of two forms:

- each encryption and blinding is annotated with a crypto-point ℓ and an assertion of the form [dest $\mathcal{L}$] meaning that the corresponding decryption or unblinding is intended to happen at one of the crypto-points mentioned in the set $\mathcal{L}$ of crypto-points.
- each decryption and unblinding operation is annotated with a crypto-point and an assertion of the form [orig $\mathcal{L}$] meaning that the value being decrypted or unblinded is intended to come from one of the crypto-points of $\mathcal{L}$.

The set $\mathcal{L}$ should of course be a subset of the entire set of crypto-points $\mathcal{C}$ occurring in the protocol. The annotations will be used in the analysis which is described in Chapter 4. It is important to note that the semantics of the annotated LySA ignores the annotations and is therfore analogues to the semantics of LySA without annotations as listed in Table A.1 in the appendix.

Chapter 3

Modelling Protocols in LySA

We are now ready to model protocols in LySA; in particular we will model the FOO92 voting protocol. The ordinary protocol narration as we presented in the introduction is given in Table 3.1 now written with the notation from LySA.

1.	V	$\rightarrow A$	:	$V,\ \{\!\{[\{v\}_r]_b\}\!\}_{K_V^-}$	Preparation Phase
2.	A	$\rightarrow V$	:	$\{\!\{[\{v\}_r]_b\}\!\}_{K_A^-}$	Administration Phase
3.	(V)	$\rightarrow C$	:	$\{\!\{\{v\}_r\}\!\}_{K_A^-}$	Voting Phase
4.	C	$\rightarrow$	:	$l,\ \{\!\{\{v\}_r\}\!\}_{K_A^-}$	Publishing Phase
5.	(V)	$\rightarrow C$	:	$l,\ r$	Opening Phase

Table 3.1: FOO92: Protocol narration

The translation from ordinary protocol narration into a LySA process is done in two stages: First we shall refine the ordinary protocol narration into an extended protocol narration, which distinguishes between inputs and corresponding outputs and also makes clear which checks must be performed. In the second stage the extended protocol narration is translated into LySA (with blinding). A discussion on the need for extending the ordinary protocol narration can be found in [1].

3.1 Extended Protocol Narration

The extended protocol narration is listed in Table 3.2 where we use the LySA terms and syntax for writing the cryptographic operations.

1.	$V\rightarrow$	:	$V, A, V, \{\!\|[\{v\}_r]_b\|\!\}_{K_V^-}$	
$1'$.	$\rightarrow A$	:	y_V, y_A, y'_V, y_1	[check $y_V = y'_V$ and $y_A = A$]
$1''$.	A	:	decrypt y_1 as $\{\!\|y_2\|\!\}_{K_{y_V}^+}$	[check V's signature]
2.	$A\rightarrow$	:	$A, y_V, \{\!\|y_2\|\!\}_{K_A^-}$	
$2'$.	$\rightarrow V$	:	x_A, x_V, x_1	[check $x_A = A$ and $x_V = V$]
$2''$.	V	:	decrypt x_1 as $\{\!\|x_2\|\!\}_{K_A^+}$	[check $x_2 = [\{v\}_r]_b$]
$2'''$.	V	:	unblind x_1 as $[\![x_3]\!]_b$	
3.	$(V)\rightarrow$	:	D, C, x_3	
$3'$.	$\rightarrow C$	:	z_D, z_C, z_1	[check $z_C = C$]
$3''$.	C	:	decrypt z_1 as $\{\!\|z_2\|\!\}_{K_A^+}$	[check A's signature]
4.	$C\rightarrow$	:	C, D, z_1, l	
$4'$.	$\rightarrow V$	:	x_C, x_D, x_4, x_5	[check $x_4 = x_3,\ x_C = C$ and $x_D = D$]
5.	$(V)\rightarrow$	:	D, C, x_5, r	
$5'$.	$\rightarrow C$	:	z'_D, z'_C, z_3, z_4	[check $z_3 = l$ and $z'_C = C$]
$5''$.	C	:	decrypt z_2 as $\{z\}_{z_4}$	

Table 3.2: FOO92: Extended protocol narration

First observe that each message is extended with source and destination information along the lines of IPv4 and IPv6. Upon receipt of a message, the principal will always check whether the message is intended for him; occasionally he will also check that the sender is who he expected. Note that these components of the message are sent in clear text and are therefore forgeable.

As mentioned earlier, we model bit commitment (message 1) as symmetric encryption with the commitment key r. Digital signatures are modelled using asymmetric encryption with the principals private key (messages 1 and 2) and verification of a signature is then modelled using asymmetric decryption with the corresponding public key (messages $1''$, $2''$ and $3''$). In addition to verifying the administrators signature (message $2''$) the voter must also ensure that the signed message was indeed his own ballot, unblinding of the signed ballot (message $2'''$) will then result in a signed commitment of the vote in accordance with the rule (Unblind 2).

Modelling the anonymous communication channel (messages 3 and 5) is done by spoofing the source with a dummy name D. The publishing of the votes

is done by sending each vote on the list to everyone on the ether - again the dummy D name can be used, now as the destination.

3.2 LySA Specification

The extended narration can be translated into LySA by dividing the narration into 3 processes, one for each principal. The LySA specification of the protocol is given in Table 3.3. Notice that the checks in the extended narration are represented by the pattern matchings on input and decryption.

As we shall see shortly the analysis of LySA does not support rebinding of variables and new variables can only be introduced by input, decryption and unblinding. Therefore a small trick has to be used when a signature has to be verified but not removed: The recipient of the message has to decrypt the signature and then resign the content by using the same signature. This does not compromise the analysis, as the signature of the message has already been verified. The trick is used in the model of both the voter and the counter (messages 3 and 4).

In the LySA specification we add annotations to all cryptographic operations as described earlier in Chapter 2. The sets of crypto-points for the destination/origin assertions depend of the property that should be analysed and we shall come back to those in Chapter 7.

In order to ensure that we analyse against the hardest attacker, the attacker should initially have knowledge of all the public keys. This is done in the LySA specification by sending these values in plain-text on the ether in parallel with the principals in the protocol.

$(\nu_{\pm}\ K_V)\,(\nu_{\pm}\ K_A)$
$($ $(\nu\, v)\,(\nu\, r)\,(\nu\, b)$ /* Voter */
1. $\langle V, A, V, \{\!|[\{v\}_r^{v_1}[\mathsf{dest}\ \mathcal{L}_{v_1}\ ]]_b^{v_2}[\mathsf{dest}\ \mathcal{L}_{v_2}\ ]|\!\}_{K_V^-}^{v_3}[\mathsf{dest}\ \mathcal{L}_{v_1}\ ]\rangle.$
$2'.$ $(A, V;\ x_1).$
$2''.$ $\mathsf{decrypt}\ x_1\ \mathsf{as}\ \{|;\ x_2|\}_{K_A^+}^{v_4}[\mathsf{orig}\ \mathcal{L}_{v_4}\ ]\ \mathsf{in}$
$2'''.$ $\mathsf{unblind}\ x_2\ \mathsf{as}\ [|;\ x_3|]_b^{v_5}[\mathsf{orig}\ \mathcal{L}_{v_5}\ ]\ \mathsf{in}$
3. $\langle D, C, \{|x_3|\}_{K_A^-}^{v_6}[\mathsf{dest}\ \mathcal{L}_{v_6}\ ]\rangle.$
$4'.$ $(C, D, \{|x_3|\}_{K_A^-}^{v_7}[\mathsf{dest}\ \mathcal{L}_{v_7}\ ];\ x_4).$
5. $\langle D, C, x_4, r\rangle.0$

$1'.$ | $(V, A, V;\ y_1).$ /* Administrator */
$1''.$ $\mathsf{decrypt}\ y_1\ \mathsf{as}\ \{|;\ y_2|\}_{K_V^+}^{a_1}[\mathsf{orig}\ \mathcal{L}_{a_1}\ ]\ \mathsf{in}$
2. $\langle A, V, \{|y_2|\}_{K_A^-}^{a_2}[\mathsf{dest}\ \mathcal{L}_{a_2}\ ]\rangle.0$

| $(\nu\, l)$ /* Counter */
$3'.$ $(D, C;\ z_1).$
$3''.$ $\mathsf{decrypt}\ z_1\ \mathsf{as}\ \{|;\ z_2|\}_{K_A^+}^{c_1}[\mathsf{orig}\ \mathcal{L}_{c_1}\ ]\ \mathsf{in}$
4. $\langle C, D, \{|z_2|\}_{K_A^-}^{c_2}[\mathsf{dest}\ \mathcal{L}_{c_2}\ ], l\rangle.$
$5'.$ $(D, C, l;\ z_3).$
$5''.$ $\mathsf{decrypt}\ z_2\ \mathsf{as}\ \{;\ z_4\}_{z_3}^{c_3}[\mathsf{orig}\ \mathcal{L}_{c_3}\ ]\ \mathsf{in}\ 0$

| $\langle K_V^+, K_A^+\rangle.0$ /* Knowledge of the attacker */
$)$

Table 3.3: FOO92 in LySA-calculus

Chapter 4

Analysis of LySA

The original analysis of LySA (without blinding) is specified as a Flow Logic in [7, 8]. In this chapter we shall present this analysis, and extend it to also include blinding.

4.1 Domain of the Analysis

The aim of the analysis is to give a safe over-approximation of all possible messages communicated on the network, along with the possible value bindings of the variables. Furthermore the analysis should record all violations that there may be to the destination/origin annotations.

In the analysis we assume perfect cryptography, meaning that decryption can only be done using the correct key, and similarly unblinding can only be done using the correct blinding factor.

The analysis of each term E will determine a superset of the possible values it may evaluate to. To do this we keep track of all potential value bindings to variables in a global abstract environment ρ:

ρ : maps the variables to all values they may be bound to.

The judgement for terms then takes the form:

$$\rho \models E : \vartheta$$

and expresses that ϑ is an acceptable estimate of the set of values that E may evaluate to in the abstract environment ρ.

In the analysis of a process P we focus on which values may flow on the network. In order to do this we keep track of all potential messages on the ether in the network component κ:

κ : includes all message tuples that may flow on the network.

To obtain this information we make use of the abstract environment ρ, and the judgement for processes has the form:

$$\rho, \kappa, \psi \models P$$

with the error component ψ:

ψ : holds an over-approximation of the origin/destination violations.

If $(\ell, \ell') \in \psi$ then something that was encrypted or blinded at crypto-point ℓ was unexpectedly decrypted or unblinded at ℓ'.

4.2 Dealing with Infinities

A LySA process may use a combination of restriction and replication to generate an arbitrarily large number of names during an execution. Hence recording all possible messages communicated on the network, would mean recording infinite sets of names.

To deal with these infinite sets in an efficiently computable way, the set Val of values, is partitioned into finitely many equivalence classes written $\lfloor \mathsf{Val} \rfloor$. This partition is made purely for the efficiency of the analysis and it is important to stress that they have no effect on the semantic behavior of a process.

Each of these equivalence classes are assigned a representative; a *canonical* value, written $\lfloor V \rfloor$. Hence, as we shall formally prove later, the analysis can only distinguish two values V_1 and V_2 if these values belong to different equivalence classes, i.e. if $\lfloor V_1 \rfloor \neq \lfloor V_2 \rfloor$. The analysis is constructed such that any "mistakes" that arise because it cannot distinguish two values, will lead to over-approximation.

4.3 Control Flow Analysis of LySA with Blinding

We are now ready to present the analysis. The clauses defining the judgements for the original LySA-calculus may be found in [7, 8], and in this section we shall extend these to include the blinding construct. The extension to the control flow analysis is presented in Table 4.1, however for the interested reader we have listed the full analysis of LySA with blinding in Table A.5 and A.6 in the appendix.

(ABli) $\rho \models [\![E_1, \cdots, E_k]\!]^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}] : \vartheta$
iff $\wedge_{i=0}^{k}\, \rho \models E_i : \vartheta_i \wedge$
$\forall\, V_0 \in \vartheta_0 \ldots V_k \in \vartheta_k : [\![V_1, \cdots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta$

(AUBli) $\rho, \kappa, \psi \models \mathsf{unblind}\ E\ \mathsf{as}\ [\![E_1, \cdots, E_j; x_{j+1}, \cdots, x_k]\!]^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j}\, \rho \models E_i : \vartheta_i \wedge$
$(\forall [\![V_1, \cdots, V_k]\!]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}'] \in \vartheta : \wedge_{i=0}^{j}\, V_i \,\mathsf{E}\, \vartheta_i \Rightarrow$
$\wedge_{i=j+1}^{k}\, V_i \in \rho(\lfloor x_i \rfloor) \wedge\ \rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$
$\wedge$
$(\forall \{\!|[\![V_1, \cdots, V_{k'}]\!]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}']|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \vartheta :$
$j = 0 \wedge k = 1 \wedge V_0 \,\mathsf{E}\, \vartheta_0 \Rightarrow$
$\{\!|V_1, \cdots, V_{k'}|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \rho(\lfloor x_1 \rfloor) \wedge$
$\rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$

Table 4.1: Analysis of terms and processes for blinding; $\rho \models E : \vartheta$ and $\rho, \kappa, \psi \models P$.

The rule for the blinding term is very straightforward and identical to that of symmetric encryption. To produce the set ϑ, the rule for k-ary blinding finds the set ϑ_i for each term E_i, collects all k-tuples of values $(V_0, \cdots, V_k)$ taken from $\vartheta_0 \times \cdots \times \vartheta_k$ into values of the form $[\![V_1, \cdots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}]$ and requires these values to belong to ϑ.

The rule for the unblinding process consists of two parts, where the first part is similar to the rule for symmetric decryption. The analysis evaluates the sets ϑ and ϑ_i, and checks for each blinded value $\{\!|V_1, \cdots, V_k|\!\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta$ wether the values $V_0, \cdots, V_j$ are pointwise included in ϑ_i; note that also the blinding factor is checked this way. Here the *faithful membership* E for matching

ignores annotations. If the check succeeds, the values $V_{j+1}, \cdots, V_k$ are pointwise ensured to be contained in x_i, and the ψ-component must contain (ℓ, ℓ') if the destination/origin assertions might be violated.

The second part of the unblinding rule is used if the process has the form $\mathsf{unblind}\ E\ \mathsf{as}\ [\![; x_1]\!]^{\ell}_{E_0}[\mathsf{orig}\ \mathcal{L}\,]\ \mathsf{in}\ P$. For each value that is signed and blinded; that is for each $\{\!|[\![V_1, \cdots, V_{k'}]\!]^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'\,]|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\ \mathcal{L}^{sig}\,] \in \vartheta$, it is checked whether the value V_0 is included into ϑ_0, again the faithful membership E is used for matching If the check is successful then the value $\{\!|V_1, \cdots, V_{k'}|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\ \mathcal{L}^{sig}\,]$ must be contained in x_1, additionally the ψ-component must contain (ℓ, ℓ') if the destination/origin assertions might be violated.

The following example gives an idea of the workings of the analysis.

Example 4.1 Analysis of a protocol *Consider the following simple protocol with two principals A and B. In the protocol, A generates a fresh key K and sends it in clear to B along with a message m which is symmetric encrypted under the key K (at crypto-point ℓ_A). Upon receiving the messages x_K and x, B decrypts x with the key x_K (at crypto-point ℓ_B).*

$$
\begin{array}{ll}
((\nu\, m)\,(\nu\, K)\,\langle A, B, K, \{m\}^{\ell_A}_K[\mathsf{dest}\ \mathcal{L}_A\,]\rangle.0 & /*\ A\ */ \\
| & \\
(A, B;\, x_K, x).\ \mathsf{decrypt}\ x\ \mathsf{as}\ \{;\, x_m\}^{\ell_B}_{x_K}[\mathsf{orig}\ \mathcal{L}_B\,]\ \mathsf{in}\ 0) & /*\ B\ */
\end{array}
$$

The encryption at crypto-point ℓ_A is intended to be decrypted only at ℓ_B and correspondingly the decryption at ℓ_B should originate from the encryption at ℓ_A, hence we have the sets of crypto-points $\mathcal{L}_A = \{\ell_B\}$ and $\mathcal{L}_B = \{\ell_A\}$.

Now the analysis of this protocol consists a conjunction of the analysis of the two processes in parallel. The analysis of the first process gives $\langle \lfloor A \rfloor, \lfloor B \rfloor, \lfloor K \rfloor, \{\lfloor m \rfloor\}^{\ell_A}_{\lfloor K \rfloor}[\mathsf{dest}\ \mathcal{L}_A\,]\rangle \in \kappa$ as this message is sent over the network. Since no other messages are sent on the network in the protocol we have that $\kappa = \{\langle \lfloor A \rfloor, \lfloor B \rfloor, \lfloor K \rfloor, \{\lfloor m \rfloor\}^{\ell_A}_{\lfloor K \rfloor}[\mathsf{dest}\ \mathcal{L}_A\,]\rangle\}$.

The analysis of the second process yields the possible bindings of variables $\lfloor K \rfloor \in \rho(x_K)$ and $\{\lfloor m \rfloor\}^{\ell_A}_{\lfloor K \rfloor}[\mathsf{dest}\ \mathcal{L}_A\,] \in \rho(x)$, and B can now decrypt x resulting in yet another variable binding $\lfloor m \rfloor \in \rho(x_m)$. Since $\ell_B \in \mathcal{L}_A$ and $\ell_A \in \mathcal{L}_B$, there will be no violations to the assertions and hence we get that $\psi = \emptyset$. □

4.4 Soundness of the Analysis

In this section we shall prove that our analysis respects the operational semantics of LySA, more precisely we shall show that ρ and κ indeed statically predicts all possible variable bindings and all messages sent on the ether during any execution of the protocol. In addition we show that an empty ψ component guarantees that there can never be any violations to the annotations.

These results require that our analysis captures the entire behavior of the protocol, which we single out in a subject reduction lemma. Also a number of other lemmata are presented, which gives perspective on the analysis, and serves as foundation of the main results.

The following proofs utilise the fact that the semantics for the annotated LySA process, as described earlier, ignores the annotations, and are therefore analogous to the semantics for LySA without annotations.

We shall first present three invariance lemmata, which shows what the analysis cannot distinguish between. The first lemma states that the analysis only distinguish processes to the level of assignment of canonical names.

Lemma 4.1 (Invariance of canonical names) *If* $\rho, \kappa, \psi \models P$ *and* $\lfloor n \rfloor = \lfloor n' \rfloor$ *then* $\rho, \kappa, \psi \models P[n \mapsto n']$.

Proof The lemma is a direct consequence of the fact that the analysis only records canonical names. The proof proceeds straightforward by induction in the definition of the analysis with the only interesting case being the rule (AN) though it too is straightforward as $\lfloor n \rfloor = \lfloor n[n \mapsto n'] \rfloor = \lfloor n \rfloor$ □

Similar lemmata can be shown about public and private keys and variables. As previously discussed this result shows, that the analysis cannot tell values that have different canonical representative apart.

The semantics of LySA presented earlier allows free α-equivalence. However, this also allows change of bound names, which again could mean change of canonical representatives. As our analysis can only operate on a finite number of canonical representatives, we shall use disciplined α-equivalence instead.

Definition 4.2 (Disciplined α-equivalence) Two processes P_1 and P_2 are *disciplined* α-equivalent whenever $P_1 \stackrel{\alpha}{\equiv} P_2$ using the rules in Table A.3 with the extra requirement that $\lfloor n_1 \rfloor = \lfloor n_2 \rfloor$, $\lfloor m_1^- \rfloor = \lfloor m_2^- \rfloor$ and $\lfloor m_1^+ \rfloor = \lfloor m_2^+ \rfloor$.

This is obviously a subrelation of ordinary α-equivalence, but since each canonical name $\lfloor n \rfloor$ represents an equivalence class with an infinite number of names, disciplined α-equivalence is as expressive as α-equivalence. In the following, the semantics using disciplined α-equivalence will be used.

Thus, we can now state the second invariance lemma.

Lemma 4.3 (Invariance of α-equivalence) *If $\rho, \kappa, \psi \models P$ and P is disciplined α-equivalent with P' then $\rho, \kappa, \psi \models P'$.*

Proof The proof proceeds by induction in the definition of α-equivalence in Table A.3. The cases for the equivalence follow by the induction hypothesis. The remaining cases follow from Lemma 4.1 remembering that substituted names have the same canonical name as their substitute. □

Hence the analysis cannot distinguish two α-equivalent processes. α-equivalence is used by structural congruence, and actually the analysis cannot tell two structural congruent processes apart either:

Lemma 4.4 (Invariance of structural congruence) *If $\rho, \kappa, \psi \models P$ and $P \equiv P'$ then $\rho, \kappa, \psi \models P'$.*

Proof The proof proceeds by induction in the definition of $P \equiv P'$ defined in Table A.2

Cases for equivalence and congruence follow by the induction hypothesis.

Cases for parallel composition follow because logical conjunction used in the analysis is commutative and associative. Furthermore, logical conjunction has true as a neutral element and true is equivalent to the analysis of 0, which is the neutral element of parallel composition.

Case for replication Assume $\rho, \kappa, \psi \models !P$. Then the following calculation justifies that also $\rho, \kappa, \psi \models P \mid !P$:

$$\begin{array}{lll} \rho, \kappa, \psi \models !P & \text{iff} & \rho, \kappa, \psi \models P \qquad\qquad\qquad\quad (\mathsf{ARep}) \\ & \text{iff} & \rho, \kappa, \psi \models P \wedge \rho, \kappa, \psi \models P \\ & \text{iff} & \rho, \kappa, \psi \models P \wedge \rho, \kappa, \psi \models !P \quad (\mathsf{ARep}) \\ & \text{iff} & \rho, \kappa, \psi \models P \mid !P \qquad\qquad\quad (\mathsf{APar}) \end{array}$$

Cases for restriction are straightforward to check, using the fact that the analysis ignores restriction.

Case for α-equivalence follows from Lemma 4.3. □

These invariance results should serve as an insight into how the analysis is restricted, such that it is efficiently computational.

It is now convenient to prove the following three lemmata. The first one shows, that the estimates of terms the analysis finds, does in fact capture all values a term may evaluate to. The next two show, that these estimates are also resistant to substitution of values for variables, and that this holds for both terms and processes.

Lemma 4.5 (Evaluation of values) *The analysis $\rho \models V : \vartheta$ holds if and only if $\lfloor V \rfloor \in \vartheta$.*

Proof The proof is by induction in the structure of values. Remembering that the values, V, are terms without variables, the proof is straightforward. □

Hence we can now show that the estimates are resistant to substitution in terms.

Lemma 4.6 (Substitution in terms) *If $\rho \models E : \vartheta$ and $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$ then $\rho \models E[x \overset{\alpha}{\mapsto} V] : \vartheta$.*

Proof The proof proceeds by structural induction over terms by regarding each of the rules in the analysis. Whenever one has to do a proof that concerns substitution, the only interesting cases are the ones where the substitution modifies something. In this proof the interesting case, thus, is (AVar). The remaining cases are straightforward as e.g.

Case (ANum). Assume that $\rho \models n : \vartheta$, For arbitrary choices of x and V it holds that $n[x \overset{\alpha}{\mapsto} V] = n$ so it is immediate that also $\rho \models n[x \overset{\alpha}{\mapsto} V] : \vartheta$.

Case (ANp), (ANm) are similar.

Case (AVar). Assume that $\rho \models x' : \vartheta$ i.e. that $\rho(\lfloor x' \rfloor) \subseteq \vartheta$. Then there are two cases. Either $x' \neq x$ in which case $x'[x \overset{\alpha}{\mapsto} V] = x'$ so clearly $\rho \models x'[x \overset{\alpha}{\mapsto} V] : \vartheta$. Alternatively, $x' = x$ in which case $x'[x \overset{\alpha}{\mapsto} V] = V$. Furthermore assume that $\lfloor V \rfloor \in \rho(\lfloor x' \rfloor)$ and in which case $\lfloor V \rfloor \in \vartheta$ by the analysis. Finally, using Lemma 4.5 one may conclude that $\rho \models V : \vartheta$ as required.

Case (ASEnc), (AAEnc), (ABli) follow directly using the induction hypothesis. □

The following Lemma shows that this also applies for processes.

Lemma 4.7 (Substitution in processes) *If* $\rho, \kappa, \psi \models P$ *and* $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$ *then* $\rho, \kappa, \psi \models P[x \overset{\alpha}{\mapsto} V]$.

Proof The proof is done by straightforward induction applying the induction hypothesis on any subprocesses and Lemma 4.6 on any subterm. It relies on Lemma 4.3 because the analysis is invariant under any disciplined α-renaming that may occur due to capture avoiding substitution. □

We are now ready to present the subject reduction result. As already explained, this lemma shows that the analysis captures any behavior of the protocol; that is that if the analysis components are an acceptable estimate for a process P then they are also an acceptable estimate for any process P' that P may evolve into.

Lemma 4.8 (Subject Reduction) *If* $\rho, \kappa, \psi \models P$ *and* $P \rightarrow P'$ *then* $\rho, \kappa, \psi \models P'$.

Proof The proof proceeds by structural induction in the reduction steps.

Case (Com). Let $P = \langle V_1, \ldots, V_k \rangle.P_1 \mid (V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k).P_2$ and $P' = P_1 \mid P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$ and assume that $\rho, \kappa, \psi \models P$ and $P \rightarrow P'$ due to (Com). Expanding the analysis one gets

$$\begin{array}{lll} \rho, \kappa, \psi \models P & \text{iff} & \rho, \kappa, \psi \models \langle V_1, \ldots, V_k \rangle.P_1 \wedge \\ & & \rho, \kappa, \psi \models (V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k).P_2 \\ & \text{iff} & \wedge_{i=1}^{k} \rho \models V_i : \vartheta_i \wedge \forall U_1 \in \vartheta_1 \ldots U_k \in \vartheta_k : U_1 \ldots U_k \in \kappa \wedge \\ & & \rho, \kappa, \psi \models P_1 \wedge \\ & & \wedge_{i=1}^{j} \rho \models V_i : \vartheta'_i \wedge \forall U'_1 \ldots U'_k \in \kappa : \wedge_{i=1}^{j} U'_i \in \vartheta'_i \Rightarrow \\ & & \quad (\wedge_{i=j+1}^{k} U'_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P_2) \end{array}$$

From the analysis of output and Lemma 4.5 one may conclude that $\rho, \kappa, \psi \models P_1$ and that κ contains $\lfloor V_1 \rfloor \ldots \lfloor V_j \rfloor \lfloor V_{j+1} \rfloor \ldots \lfloor V_k \rfloor$. Using the latter, the analysis of the input and Lemma 4.5 furthermore gives that $\lfloor V_i \rfloor \in \rho(\lfloor x_i \rfloor)$ for $i = j+1, \ldots, k$ and that $\rho, \kappa, \psi \models P_2$. By repeatedly applying Lemma 4.7, one may conclude that $\rho, \kappa, \psi \models P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$ i.e. that $\rho, \kappa, \psi \models P'$ as required.

Case (UBli1). Let

$$P = \mathsf{unblind}\ [\![V_1, \ldots, V_k]\!]^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'\,]\ \mathsf{as}\ [\![V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k]\!]^{\ell}_{V_0}[\mathsf{orig}\ \mathcal{L}\,]\ \mathsf{in}\ P''$$

and $P' = P''[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$. Assume that $\rho, \kappa, \psi \models P$ and that $P \rightarrow P'$ because of (UBli1). From the definition of the analysis using the rule

(ABli) and Lemma 4.5 it is clear that $[\![\lfloor V_1 \rfloor, \ldots, \lfloor V_k \rfloor]\!]^{\ell'}_{\lfloor V_0 \rfloor}[\mathsf{dest}\,\mathcal{L}']\in\vartheta$ and that $\lfloor V_i\rfloor\in\vartheta_i$ for $i=0,\ldots,j$. Thus, the analysis of pattern matching in the first part (AUBli) succeeds and one may conclude that $\lfloor V_i\rfloor\in\rho(\lfloor x_i\rfloor)$ for $i=j+1,\ldots,k$ and that $\rho,\kappa,\psi\models P''$. Lemma 4.7 then gives that also $\rho,\kappa,\psi\models P'$

Case (UBli2). Let

$$P=\mathsf{unblind}\ \{\!|[\![V_1,\cdots,V_{k'}]\!]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}']|\!\}^{\ell^{sig}}_{m^-}[\mathsf{dest}\,\mathcal{L}^{sig}]\ \mathsf{as}\ [\![;x]\!]^{\ell}_{V_0}[\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P''$$

and $P'=P''[x\overset{\alpha}{\mapsto}\{\!|V_1,\ldots,V_k|\!\}^{\ell^{sig}}_{m^-}[\mathsf{dest}\,\mathcal{L}^{sig}]]$. Assume that $\rho,\kappa,\psi\models P$ and that $P\to P'$ because of (UBli2). From rule (ABli) in the definition of the analysis and Lemma 4.5 it is clear that $\{\!|[\![\lfloor V_1\rfloor,\cdots,\lfloor V_{k'}\rfloor]\!]^{\ell'}_{\lfloor V_0\rfloor}[\mathsf{dest}\,\mathcal{L}']|\!\}^{\ell^{sig}}_{\lfloor m^-\rfloor}[\mathsf{dest}\,\mathcal{L}^{sig}]\in\vartheta$ and that $\lfloor V_0\rfloor\in\vartheta_0$. Thus, the analysis of pattern matching in the second part (AUBli) succeeds and one may conclude that $\{\!|\lfloor V_1\rfloor,\ldots,\lfloor V_k\rfloor|\!\}^{\ell^{sig}}_{\lfloor m^-\rfloor}[\mathsf{dest}\,\mathcal{L}^{sig}]\in\rho(\lfloor x\rfloor)$ and that $\rho,\kappa,\psi\models P''$. Lemma 4.7 then gives that also $\rho,\kappa,\psi\models P'$

Case (SDec), (ADec), (ASig) are similar.

Case (New). Assume $\rho,\kappa,\psi\models(\nu\,n)\,P$ i.e. that $\rho,\kappa,\psi\models P$. Assume also that $(\nu\,n)\,P\to(\nu\,n)\,P'$ using (New) because $P\to P'$. Then by the induction hypothesis $\rho,\kappa,\psi\models P'$, which by the analysis definition allows us to conclude that $\rho,\kappa,\psi\models(\nu\,n)\,P'$.

Case (ANew) is similar.

Case (Par) Assume that $\rho,\kappa,\psi\models P_1\mid P_2$ i.e. that $\rho,\kappa,\psi\models P_1$ and $\rho,\kappa,\psi\models P_2$. Furthermore assume that $P_1\mid P_2\to P_1'\mid P_2$ by (Par) because $P_1\to P_1'$. Then using the induction hypothesis also (Par) $\rho,\kappa,\psi\models P_1'$. The analysis then allows to conclude that $\rho,\kappa,\psi\models P_1'\mid P_2$.

Case (Congr) is a direct consequence of the induction hypothesis and application of Lemma 4.4. □

Using the subject reduction result, we can now present our main results. First we show that any message that may flow on the network will be in κ if $\rho,\kappa,\psi\models P$.

Theorem 4.9 (Messages in κ) *If $\rho,\kappa,\psi\models P$ and $P\to^* P'\to P''$ such that the reduction $P'\to P''$ is derived using* (Com) *on output $\langle V_1,\ldots,V_k\rangle.P_1''$ then $\lfloor V_1\rfloor\ldots\lfloor V_k\rfloor\in\kappa$.*

Proof By induction in the length of the reduction sequence, Lemma 4.8 can be used to conclude that $\rho,\kappa,\psi\models P'$. Next the proof proceeds by induction in

the reduction rules used to derive $P' \rightarrow P''$.

Case (Com) If this rule is applied, it will be a process of the form

$$\langle V_1, \ldots, V_k \rangle.P_1'' \mid (V_1, \ldots, V_j; x_{j+1}, \ldots, x_k).P_2''$$

The analysis holds for this process meaning, in particular, that the analysis of output holds for the communication of the k-tuple. Using Lemma 4.5 one can check that then indeed $\lfloor V_1 \rfloor \ldots \lfloor V_k \rfloor \in \kappa$.

Case (SDec), (ADec), (ASig), (UBli1), (UBli2). Reductions that uses any of these rules will not also use the rule (Com) and can therefore be disregarded.

Case (New), (ANew), (Par), (Congr) are all straightforward by applying the induction hypothesis noting that the analysis also holds for any subprocesses. □

And we present a similar result for all possible variable bindings ρ.

Theorem 4.10 (Values in ρ) *If $\rho, \kappa, \psi \models P$ and $P \rightarrow^* P' \rightarrow P''$ such that P'' is the result of a substitution of the variable x for the value V then $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$.*

Proof The proof is similar to that of Theorem 4.9. □

Which leads us to our last main result; if there exist a κ and a ρ such that $\rho, \kappa, \emptyset \models P$ for a protocol P, then we are certain that no evaluation of P will ever lead to a violation of the annotations. This is shown by the following lemma.

Theorem 4.11 (Analysis of authentication) *If $\rho, \kappa, \psi \models P$ and $\psi = \emptyset$ then P ensures destination and origin authentication.*

Proof The theorem can be proven by showing an extended subject reduction result that says if $\rho, \kappa, \psi \models P$ and $P \rightarrow P'$ then $\rho, \kappa, \psi \models P'$ and furthermore if $\psi = \emptyset$ then $P \rightarrow P'$ does not violate the authentication property. An induction in the length of the execution sequence then gives that P ensures authentication for all executions.

The interesting part of the proof is the cases for decryption and unblinding. One of these is given below; the others are analogous.

Case (UBli1). Let P be the process

$$\mathsf{unblind}\ [\![V_1, \ldots, V_k]\!]^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'\,]\ \mathsf{as}\ [\![V_1, \ldots, V_j;\ x_{j+1}, \ldots, x_k]\!]^{\ell}_{V_0}[\mathsf{orig}\ \mathcal{L}\,]\ \mathsf{in}\ P''$$

and assume that $\rho, \kappa, \psi \models P$ and that $P \to P'$ by (UBli1). The argument that then also $\rho, \kappa, \psi \models P'$ follows directly from Theorem 4.8. As the analysis of the pattern matching succeeds, in particular for the encrypted value that is decrypted in P, the analysis $\rho, \kappa, \psi \models P'$ gives that

$$\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi$$

holds. Next, assume that $\psi = \emptyset$. Then the above part of the analysis gives that $\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow \mathsf{false}$, which can only hold if

$$\neg(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L})$$

This directly says that the authentication property will not be violated by the reduction $P \to P'$. □

Chapter 5

The Attacker

Protocols that involve several principals who communicate on a network, are vulnerable to attacks from other parties, that have access to the network. In LySA we have one global network; the ether, and in this setting any attacker can be modelled by

$$P \mid P_\bullet$$

where P is the protocol and $P_\bullet$ represents some arbitrary attacker. This attacker can see all messages sent on the ether, but do not have access to any of the internal values in the protocol; ie. restricted names. This essentially describes the scenario of the attacker formalised by Dolev and Yao in 1981 [16].

Given a protocol P and a specific attacker $P_\bullet$, the analysis presented in Chapter 4 can account for the behavior of P under attack from $P_\bullet$. However, in order to be sure that we analyse a protocol against any attacker, we need to capture all the capabilities an attacker may have in our definition of the attacker. In this chapter we will define such an attacker and establish the correctness of this definition.

5.1 Modelling the Attacker

We aim at finding a formula $\mathcal{F}^{\mathsf{DY}}$ which characterises all attackers. This means that if (ρ, κ, ψ) satisfies $\mathcal{F}^{\mathsf{DY}}$, then $\rho, \kappa, \psi \models P_\bullet$ for all attackers $P_\bullet$.

Our approach to finding such a formula, requires a few assumptions that benefit the control flow analysis but have no semantic consequence. In [7] it is shown that a process P can be typed to $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ whenever (1) it has no free variables, (2) its free names are in $\mathcal{N}_\mathsf{f}$, (3) all arities used for sending or receiving are in $\mathcal{A}_\kappa$ and (4) all the arities used for symmetric encryption or decryption are in $\mathcal{A}_{\mathsf{Enc}}$. In our analysis however, we also use asymmetric encryption and blinding, so we extend this typing of a process to $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}}, \mathcal{A}_{\mathsf{AEnc}}, \mathcal{A}_{\mathsf{Bli}})$ whenever (5) all the arities used for asymmetric encryption or decryption are in $\mathcal{A}_{\mathsf{AEnc}}$ and (6) all the arities used for symmetric encryption or decryption are in $\mathcal{A}_{\mathsf{Bli}}$. In order to avoid special cases we shall assume that $\mathcal{A}_\kappa$, $\mathcal{A}_{\mathsf{Enc}}$, $\mathcal{A}_{\mathsf{AEnc}}$ and $\mathcal{A}_{\mathsf{Bli}}$ all contain at least one integer, eg. 1.

Given a set $\mathcal{A}_{\mathsf{Enc}}$ we now define $\mathcal{A}^+_{\mathsf{Enc}} = \mathcal{A}_{\mathsf{Enc}} \cup k^+_{\mathsf{Enc}}$ where $k^+_{\mathsf{Enc}} > 0$ and $k^+_{\mathsf{Enc}} \notin \mathcal{A}_{\mathsf{Enc}}$. Using an analogous definition of $\mathcal{A}^+_{\mathsf{AEnc}}$ and $\mathcal{A}^+_{\mathsf{Bli}}$ we now claim that attackers can be typed as $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ and that this typing does not cause any loss of generality. In particular we claim the ability of the attacker to use:

- additional free names, may be masked by restricting the names as to become local within the attacker,
- k-ary communication for $k \notin \mathcal{A}_\kappa$ does not increase his computational power,
- symmetric or asymmetric encryption and decryption of k-ary sequences where $k \notin \mathcal{A}^+_{\mathsf{Enc}}$ or $k \notin \mathcal{A}^+_{\mathsf{AEnc}}$ respectively, will not increase his computational power, and
- blinding or unblinding of k-ary sequences for $k \notin \mathcal{A}^+_{\mathsf{Bli}}$ will not increase his computational power.

An arbitrary attacker may use any canonical name or variable, but the formula $\mathcal{F}^{\mathsf{DY}}$ needs to have only a finite set of canonical names and variables. We therefore introduce canonical names $n_\bullet$, $m^+_\bullet$ and $m^-_\bullet$ in which all canonical names, public keys and private keys of the attacker are coalesced into. Similarly we introduce a canonical variable $z_\bullet$ in which all variables of the attacker are coalesced into.

A last aspect of the attacker is the annotations. In theory the attacker should not have any annotations, as annotations should only express the intentions of the protocol. However, as the syntax of LySA enforces annotations, we replace all crypto-points in the attacker by a crypto-point $\ell_\bullet$ and annotate with the trivial assertions $[\mathsf{orig}\ \mathcal{C}]$ and $[\mathsf{dest}\ \mathcal{C}]$, where the set $\mathcal{C}$ is the entire set of crypto-points occurring in the protocol including the attackers own crypto-point $\ell_\bullet$.

We can now express the capabilities of the attacker. In the style of the Dolev-Yao attacker [16], the attacker can perform the following actions:

- receive all messages sent on the ether,
- decrypt messages if he knows the key or unblind messages if he knows the blinding factor,
- construct new encryptions or blindings from values he knows,
- send messages constructed from values he knows, and
- generate new values.

The definition of the formula $\mathcal{F}^{\mathsf{DY}}$ of type $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ for expressing the Dolev-Yao condition for the extended LySA, is given as a conjunction of the 10 components in Table 5.1.

The formula shows that the attacker initially has some knowledge ($\mathsf{DY1}$), that he will learn everything sent on the ether ($\mathsf{DY2}$), that he can construct new composite values from known values using encryption and blinding ($\mathsf{DY3} - \mathsf{DY5}$), that he can decrypt using known keys ($\mathsf{DY6}, \mathsf{DY7}$), that he can unblind using known blinding factors ($\mathsf{DY8}, \mathsf{DY9}$) and that he may forge new communications ($\mathsf{DY10}$). Note that, since $\ell \in \mathcal{C}$ is always satisfied, $(\ell \notin \mathcal{C} \vee \ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi)$ can be written by the more compact $(\ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi)$.

Lets look at the protocol from Example 4.1 again, this gives an idea of how the analysis works when we analyse a protocol in parallel with the attacker.

Example 5.1 Analysis of a protocol (Continued) *Consider again the protocol from Example 4.1.*

$$\begin{array}{ll} ((\nu\, m)\,(\nu\, K)\,\langle A, B, K, \{m\}^{\ell_A}_K\,[\mathsf{dest}\ \mathcal{L}_A\,]\rangle.0 & /*\, A\, */ \\ \mid & \\ (A, B;\, x_K, x).\ \mathsf{decrypt}\ x\ \mathsf{as}\ \{;\, x_m\}^{\ell_B}_{x_K}[\mathsf{orig}\ \mathcal{L}_B\,]\ \mathsf{in}\ 0) & /*\, B\, */ \end{array}$$

Where $\mathcal{L}_A = \{\ell_B\}$ *and* $\mathcal{L}_B = \{\ell_A\}$.

$$
\begin{array}{ll}
\text{(DY1)} & \{n_\bullet, m_\bullet^+, m_\bullet^-\} \cup \lfloor \mathcal{N}_\mathsf{f} \rfloor \subseteq \rho(z_\bullet) \\
\text{(DY2)} & \wedge_{k \in \mathcal{A}_\kappa} \quad \forall \langle V_1, \cdots, V_k \rangle \in \kappa : \wedge_{i=1}^k V_i \in \rho(z_\bullet) \\
\text{(DY3)} & \wedge_{k \in \mathcal{A}_{\mathsf{Enc}}^+} \quad \forall V_0, \cdots, V_k : \wedge_{i=0}^k V_i \in \rho(z_\bullet) \Rightarrow \{V_1, \cdots, V_k\}_{V_0}^{\ell_\bullet}[\mathsf{dest}\,\mathcal{C}\,] \in \rho(z_\bullet) \\
\text{(DY4)} & \wedge_{k \in \mathcal{A}_{\mathsf{AEnc}}^+} \forall V_0, \cdots, V_k : \wedge_{i=0}^k V_i \in \rho(z_\bullet) \Rightarrow \{\!|V_1, \cdots, V_k|\!\}_{V_0}^{\ell_\bullet}[\mathsf{dest}\,\mathcal{C}\,] \in \rho(z_\bullet) \\
\text{(DY5)} & \wedge_{k \in \mathcal{A}_{\mathsf{Bli}}^+} \quad \forall V_0, \cdots, V_k : \wedge_{i=0}^k V_i \in \rho(z_\bullet) \Rightarrow [V_1, \cdots, V_k]_{V_0}^{\ell_\bullet}[\mathsf{dest}\,\mathcal{C}\,] \in \rho(z_\bullet) \\
\text{(DY6)} & \wedge_{k \in \mathcal{A}_{\mathsf{Enc}}^+} \quad \forall \{V_1, \cdots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,] \in \rho(z_\bullet) : V_0 \in \rho(z_\bullet) \Rightarrow \\
& \quad \wedge_{i=1}^k V_i \in \rho(z_\bullet) \wedge \\
& \quad (\ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi) \\
\text{(DY7)} & \wedge_{k \in \mathcal{A}_{\mathsf{AEnc}}^+} \forall \{\!|V_1, \cdots, V_k|\!\}_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,] \in \rho(z_\bullet) : \\
& \quad \forall (m^+, m^-) : \forall V_0' \in \rho(z_\bullet) : \{V_0, V_0'\} = \{m^+, m^-\} \Rightarrow \\
& \qquad \wedge_{i=1}^k V_i \in \rho(z_\bullet) \wedge \\
& \qquad (\ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi) \\
\text{(DY8)} & \wedge_{k \in \mathcal{A}_{\mathsf{Bli}}^+} \quad \forall [V_1, \cdots, V_k]_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,] \in \rho(z_\bullet) : V_0 \in \rho(z_\bullet) \Rightarrow \\
& \quad \wedge_{i=1}^k V_i \in \rho(z_\bullet) \wedge \\
& \quad (\ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi) \\
\text{(DY9)} & \wedge_{k \in \mathcal{A}_{\mathsf{Bli}}^+} \quad \forall \{\!|[V_1, \cdots, V_k]_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,]|\!\}_{V_0^{sig}}^{\ell^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \rho(z_\bullet) : V_0 \in \rho(z_\bullet) \Rightarrow \\
& \quad \{\!|V_1, \cdots, V_k|\!\}_{V_0^{sig}}^{\ell^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \rho(z_\bullet) \wedge \\
& \quad (\ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi) \\
\text{(DY10)} & \wedge_{k \in \mathcal{A}_\kappa} \quad \forall V_1, \cdots, V_k : \wedge_{i=1}^k V_i \in \rho(z_\bullet) \Rightarrow \langle V_1, \cdots, V_k \rangle \in \kappa
\end{array}
$$

Table 5.1: The Dolev-Yao attacker in extended LySA.

The analysis of this protocol gives $\langle \lfloor A \rfloor, \lfloor B \rfloor, \lfloor K \rfloor, \{\lfloor m \rfloor\}_{\lfloor K \rfloor}^{\ell_A}[\mathsf{dest}\,\mathcal{L}_A\,] \rangle \in \kappa$ *as this message is sent over the network. Since the attacker learns everything sent on the ether, the analysis also gives* $\lfloor K \rfloor \in \rho(z_\bullet)$ *as well as* $\{\lfloor m \rfloor\}_{\lfloor K \rfloor}^{\ell_A}[\mathsf{dest}\,\mathcal{L}_A\,] \in \rho(z_\bullet)$*. Since the attacker knows the key* K*, he can now decrypt the message* $\{m\}_K^{\ell_A}[\mathsf{dest}\,\mathcal{L}_A\,]$ *resulting in the violation to the annotations* $(\ell_A, \ell_\bullet) \in \psi$*. The analysis also gives the violation* $(\ell_\bullet, \ell_B) \in \psi$ *as* B *does not know the key in advance, therefore the attacker can create a new key* $K_\bullet$ *and a new message* $m_\bullet$ *and send the message* $\langle A, B, K_\bullet, \{m_\bullet\}_{K_\bullet} \rangle$ *onto the ether, which would be accepted by* B*.* □

5.2 Correctness of the Attacker

We will now establish the correctness of the Dolev-Yao condition, this is split into a soundness and a completeness result. Soundness means that $\mathcal{F}^{\mathsf{DY}}$ captures the capabilities of any attacker $P_\bullet$ and completeness means that there indeed exist an attacker as strong as the formula $\mathcal{F}^{\mathsf{DY}}$.

That soundness is satisfied is captured by the following theorem:

Theorem 5.1 (Soundness of Dolev-Yao condition) *If* (ρ, κ, ψ) *satisfies* $\mathcal{F}^{\mathsf{DY}}$ *of type* $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ *then* $\rho, \kappa, \psi \models P_\bullet$ *for all processes* $P_\bullet$ *of type* $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$.

Proof A process $\overline{P_\bullet}$ is said to have extended type $(z_\bullet, \{n_\bullet, m^+_\bullet, m^-_\bullet\} \cup \mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ whenever the canonical variables are in $\{z_\bullet\}$, the canonical names are in $\{n_\bullet, m^+_\bullet, m^-_\bullet\} \cup \mathcal{N}_\mathsf{f}$, all arities for sending and receiving are in $\mathcal{A}_\kappa$, all arities used for symmetric and asymmetric encryption and decryption are in $\mathcal{A}^+_{\mathsf{Enc}}$ and $\mathcal{A}^+_{\mathsf{AEnc}}$ respectively, and all arities used for blinding and unblinding are in $\mathcal{A}^+_{\mathsf{Bli}}$.

Now using structural induction we shall prove that

> If (ρ, κ, ψ) satisfies $\mathcal{F}^{\mathsf{DY}}$ of type $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ then $\rho, \kappa, \psi \models \overline{P_\bullet}$ for all attackers $\overline{P_\bullet}$ of extended type $(z_\bullet, \{n_\bullet, m^+_\bullet, m^-_\bullet\} \cup \mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$.

The most interesting case is when the attacker $\overline{P_\bullet}$ is the process

$$\mathsf{unblind}\ \overline{E}\ \mathsf{as}\ [\![\overline{E_1}, \cdots, \overline{E_j}; x_{j+1}, \cdots, x_k]\!]^{\ell_\bullet}_{\overline{E_0}}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in}\ \overline{P}$$

and here we need to find ϑ and $\vartheta_0, \cdots, \vartheta_j$ and show

$$(a) \quad \rho \models \overline{E} : \vartheta \wedge \wedge_{i=0}^{j} \rho \models \overline{E_i} : \vartheta_i$$

and (b) for all $[\![V_1, \cdots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta$ with $\wedge_{i=0}^{j} V_i \,\mathsf{E}\, \vartheta_i$ that:

$$(b1) \quad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor)$$
$$(b2) \quad \rho, \kappa, \psi \models \overline{P}$$
$$(b3) \quad (\ell \notin \mathcal{C} \vee \ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi))$$

and (c) for all $\{\![[\![V_1, \cdots, V_{k'}]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,]]\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \vartheta$ with $j = 0 \wedge k = 1$

and $V_0 \,\mathsf{E}\, \vartheta_0$ that:

$$
\begin{array}{ll}
(c1) & \{\!|V_1, \cdots, V_{k'}|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \rho(\lfloor x_1 \rfloor) \\
(c2) & \rho, \kappa, \psi \models \overline{P} \\
(c3) & (\ell \notin \mathcal{C} \vee \ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi))
\end{array}
$$

Choosing ϑ as the least set such that $\rho \models \overline{E} : \vartheta$, we show that it must be the case that $\vartheta \subseteq \rho(z_\bullet)$; obviously, if $\overline{E}$ has free variables $z_1, \cdots, z_m$ then ϑ consist of all values $\lfloor \overline{E}[z_1 \overset{\alpha}{\mapsto} V_1, \cdots, z_m \overset{\alpha}{\mapsto} V_m] \rfloor$ where $V_i \in \rho(z_\bullet)$. Now, since this also applies for $\vartheta_0, \cdots, \vartheta_j$, we have that (a) is satisfied. Then assume that $[\![V_1, \cdots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta$ and $\wedge_{i=0}^{j} V_i \,\mathsf{E}\, \vartheta_i$. As for (a) we have that $\wedge_{i=0}^{j} \vartheta_i \subseteq \rho(z_\bullet)$, and hence that $\wedge_{i=0}^{j} V_i \,\mathsf{E}\, \rho(z_\bullet)$, and in particular that $V_0 \,\mathsf{E}\, \rho(z_\bullet)$. Then according to (DY8) in Table 5.1, it must be the case that $\wedge_{i=0}^{k} V_i \in \rho(z_\bullet)$ and $(\ell_\bullet \notin \mathcal{L} \Rightarrow (\ell, \ell_\bullet) \in \psi)$. Since $\ell \in \mathcal{C}$ is trivially true and as $\lfloor x_i \rfloor = z_\bullet$ we have that $(b1)$ and $(b3)$ are satisfied; furthermore since $\overline{P}$ also must have extended type $(z_\bullet, \{n_\bullet, m^+_\bullet, m^-_\bullet\} \cup \mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ the induction hypothesis gives us $(b2)$ and hence we have shown that (b) is satisfied. The proof for (c) is analogous to that of (b).

All remaining cases are similar. □

The next step is to show that there actually exists such an attacker, and that we did not add any abilities that a real attacker could not posses. This can be regarded as the completeness of the attacker and is captured by the following theorem:

Theorem 5.2 (Completeness of Dolev-Yao condition) *There exists an attacker* P_{Hard} *of type* $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$ *such that the formula* $\rho, \kappa, \psi \models P_{\mathsf{Hard}}$ *is equivalent to the formula* $\mathcal{F}^{\mathsf{DY}}$ *of type* $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$.

Proof P_{Hard} is $!(P_1 \;||_{k \in \mathcal{A}_\kappa} P_2^k \;||_{k \in \mathcal{A}^+_{\mathsf{Enc}}} P_3^k \;||_{k \in \mathcal{A}^+_{\mathsf{AEnc}}} P_4^k \;||_{k \in \mathcal{A}^+_{\mathsf{Bli}}} P_5^k \;||_{k \in \mathcal{A}^+_{\mathsf{Enc}}} P_6^k \;||_{k \in \mathcal{A}^+_{\mathsf{AEnc}}} P_7^k \;||_{k \in \mathcal{A}^+_{\mathsf{Bli}}} P_8^k \mid P_9 \;||_{k \in \mathcal{A}_\kappa} P_{10}^k)$ where P_i^k corresponds to the i-th component of $\mathcal{F}^{\mathsf{DY}}$ in Table 5.1. Assuming that all variables $z, z_0, z_1, \cdots,$ have

canonical representative $z_\bullet$ and $1 \in A_\kappa$ as discussed above we have:

$$
\begin{array}{lcl}
P_1 & = & \langle n_\bullet \rangle.\langle m_\bullet^+ \rangle.\langle m_\bullet^- \rangle.0 \mid\mid_{n \in \mathcal{N}_\mathsf{f}} \langle n \rangle.0 \\
P_2^k & = & (;\, z_1, \cdots, z_k).0 \\
P_3^k & = & (;\, z_0).\cdots(;\, z_k).\langle \{ z_1, \cdots, z_k \}_{z_0}^{\ell_\bullet} [\mathsf{dest}\, \mathcal{C}\,] \rangle.0 \\
P_4^k & = & (;\, z_0).\cdots(;\, z_k).\langle \{\!| z_1, \cdots, z_k |\!\}_{z_0}^{\ell_\bullet} [\mathsf{dest}\, \mathcal{C}\,] \rangle.0 \\
P_5^k & = & (;\, z_0).\cdots(;\, z_k).\langle [\![z_1, \cdots, z_k]\!]_{z_0}^{\ell_\bullet} [\mathsf{dest}\, \mathcal{C}\,] \rangle.0 \\
P_6^k & = & (;\, z).(;\, z_0).\mathsf{decrypt}\ z\ \mathsf{as}\ \{;\, z_1, \cdots, z_k \}_{z_0}^{\ell_\bullet} [\mathsf{orig}\, \mathcal{C}\,]\ \mathsf{in}\ 0 \\
P_7^k & = & (;\, z).(;\, z_0).\mathsf{decrypt}\ z\ \mathsf{as}\ \{\!|;\, z_1, \cdots, z_k |\!\}_{z_0}^{\ell_\bullet} [\mathsf{orig}\, \mathcal{C}\,]\ \mathsf{in}\ 0 \\
P_8^k & = & (;\, z).(;\, z_0).\mathsf{unblind}\ z\ \mathsf{as}\ [\![;\, z_1, \cdots, z_k]\!]_{z_0}^{\ell_\bullet} [\mathsf{orig}\, \mathcal{C}\,]\ \mathsf{in}\ 0 \\
P_9 & = & (;\, z).(;\, z_0).\mathsf{unblind}\ z\ \mathsf{as}\ [\![;\, z_1]\!]_{z_0}^{\ell_\bullet} [\mathsf{orig}\, \mathcal{C}\,]\ \mathsf{in}\ 0 \\
P_{10}^k & = & (;\, z_1).\cdots(;\, z_k).\langle z_1, \cdots, z_k \rangle.0
\end{array}
$$

□

This also shows that the Dolev-Yao attacker is the so called *hardest attacker* [32] and concludes the proof for the correctness of the attacker.

5.3 Crypto-based Authentication

The annotations of LySA are designed to validate destination and origin authentication. More formally, we say that a process P guarantees *static authentication* if there exist ρ and κ such that $\rho, \kappa, \emptyset \models P$ and $(\rho, \kappa, \emptyset)$ satisfies $\mathcal{F}^{\mathsf{DY}}$. Now what we actually want is to validate, is wether the process P guarantees *dynamic authentication*; that $P|P_\bullet$ ensures destination and origin authentication for any attacker $P_\bullet$.

Hence we are now ready to state the main theorem:

Theorem 5.3 (Authentication) *If P guarantees static authentication then P guarantees dynamic authentication.*

Proof If $\rho, \kappa, \emptyset \models P$ and $(\rho, \kappa, \emptyset)$ satisfies $\mathcal{F}^{\mathsf{DY}}$ then, according to Theorem 4.11, it must also be the case that $\rho, \kappa, \emptyset \models P_\bullet$ for any attacker $P_\bullet$. Now using the rule (APar) in Table A.6 we get that $\rho, \kappa, \emptyset \models P \mid P_\bullet$ for any $P_\bullet$, which according to Theorem 5.1 states that $P|P_\bullet$ ensures destination and origin authentication for any attacker $P_\bullet$. This concludes the proof. □

Chapter 6

Implementation

The aim of the implementation is to compute a least (ρ, κ, ψ) for any given LySA -process P, such that $\rho, \kappa, \psi \models P$. However, as the analysis components ρ, κ and ψ are interpreted over an infinite universe of terms, this poses a termination problem for the implementation. In LySA this challenge is handled by encoding sets of terms using *tree grammars*; thereby describing the infinite sets using a finite number of grammar rules.

The implementation is carried out in two steps; first the analysis is encoded using a suitable constraint language and then this constraint is solved using a constraint solver. As developing a constraint solver is a challenging project by itself, the easiest way to do this is using an already developed solver, and in LySA the choice of solver has been the *Succinct Solver* [33]. The input to the Succinct Solver is a formula in *Alternation-free Least Fixed Point logic* (ALFP). When the ALFP formula is interpreted over a finite, structured universe, the Succinct Solver will compute the least interpretation of predicate symbols that satisfy the formula and guarantee termination. This fits well with the encoding of infinite sets as tree grammars.

The transformation from the initial LySA process P to ALFP is done by transforming the analysis presented in Chapter 4 into a *generator function* $\mathcal{G}$ for ALFP. This transformation is done in a number of steps:

Step 0 The initial step

Step 1 Transformation from Flow Logic to Verbose.

Step 2 From infinite to finite

Step 3 Removing polyvariance

Step 4 Generating ALFP

Each of these steps are described in detail for the original LySA in [9], where the implementation is also shown to be sound with respect to the analysis; that is it produces a solution which is also a solution to the analysis.

In this section we will describe how the extension of blinding can be implemented, following the same strategy, and then prove that this extension is sound with respect to the analysis. Finally we will describe how the Dolev-Yao attacker presented in Chapter 5 can be implemented as well, and show that also this implementation is sound with respect to the definition of the attacker.

6.1 Step 0 - The Initial Step

The original analysis, called the *succinct* flow analysis [35], was presented in Chapter 4. As a reminder the analysis of blinding is relisted in Table 6.2. This analysis uses the the three components ρ, κ and ψ which are summarised in Table 6.1.

ϑ	$\in$	$\wp(\mathcal{V})$	Local term cache
ρ	$:$	$\lfloor\mathcal{X}\rfloor \to \wp(\mathcal{V})$	Variable environment
κ	$\in$	$\wp(\mathcal{V}^*)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
V	$\in$	$\mathcal{V}$	Values - terms without variables
x	$\in$	$\mathcal{X}$	Variables
ℓ	$\in$	$\mathcal{C}$	Crypto-points

Table 6.1: Components used in the original analysis

In the rest of this chapter, this analysis will be transformed in order to provide the ALFP formulae. To distinguish each step, the judgement symbol $\models$ as well as as some of the analysis components will be decorated with different superscripts.

(ABli) $\rho \models [E_1, \cdots, E_k]^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}] : \vartheta$
iff $\wedge_{i=0}^{k}\, \rho \models E_i : \vartheta_i \wedge$
$\forall\, V_0 \in \vartheta_0 \ldots V_k \in \vartheta_k : [V_1, \cdots, V_k]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta$

(AUBli) $\rho, \kappa, \psi \models \mathsf{unblind}\ E\ \mathsf{as}\ [E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k]^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j}\, \rho \models E_i : \vartheta_i \wedge$
$(\forall [V_1, \cdots, V_k]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}'] \in \vartheta : \wedge_{i=0}^{j}\, V_i \,\mathsf{E}\, \vartheta_i \Rightarrow$
$\wedge_{i=j+1}^{k}\, V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$
$\wedge$
$(\forall \{\!| [V_1, \cdots, V_{k'}]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}'] |\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \vartheta :$
$j = 0 \wedge k = 1 \wedge V_0 \,\mathsf{E}\, \vartheta_0 \Rightarrow$
$\{\!| V_1, \cdots, V_{k'} |\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \rho(\lfloor x_1 \rfloor) \wedge$
$\rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$

Table 6.2: Step 0 - Annotated LySA in succinct Flow Logic

6.2 Step 1 - From Flow Logic to Verbose

The analysis in Table 6.2 is called succinct because of the succinct element ϑ. However, ALFP does not provide scoping mechanisms, and therefore the analysis must be rewritten to use only global components. This can be achieved by transforming the flow logic into *verbose* form [35], where all components are placed on the left hand side of the iff.

Transforming a succinct flow logic into verbose is done in the standard way [35]: We add labels to all places where succinct judgements are used and then replace each succinct component with a global component which maps each label to the corresponding succinct component. For the LySA analysis this means adding labels, $l \in Lab$, to terms, yielding the new syntax:

$$E ::= \ldots \mid [\![E_1^{l_1}, \cdots, E_k^{l_k}]\!]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\,\mathcal{L}]^{l}$$
$$P ::= \ldots \mid \mathsf{unblind}\ E^{l}\ \mathsf{as}\ [\![E_1^{l_1}, \cdots, E_j^{l_j};\, x_{j+1}, \cdots, x_k]\!]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$$

The components for the verbose analysis are therefore similar to those of the succinct analysis, except that we now have labels and the succinct element ϑ is

replaced by a mapping ϑ^v from labels to the local cache. The components are presented in Table 6.3.

ϑ^v	:	$Lab \to \wp(\mathcal{V})$	Global term cache
ρ	:	$\lfloor \mathcal{X} \rfloor \to \wp(\mathcal{V})$	Variable environment
κ	$\in$	$\wp(\mathcal{V}^*)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
V	$\in$	$\mathcal{V}$	Values - terms without variables
x	$\in$	$\mathcal{X}$	Variables
ℓ	$\in$	$\mathcal{C}$	Crypto-points
l	$\in$	Lab	Labels on terms

Table 6.3: Components used in the verbose analysis

The resulting analysis from applying these modifications is listed in Table 6.4.

(VBli) $\rho, \vartheta^v \models^v [E_1^{l_1}, \cdots, E_k^{l_k}]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\,\mathcal{L}]^l$

iff $\wedge_{i=0}^{k} \rho, \vartheta^v \models^v E_i^{l_i} \wedge$
$\forall V_0 \in \vartheta^v(l_0) \ldots V_k \in \vartheta^v(l_k) : [V_1, \cdots, V_k]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta^v(l)$

(VUBli) $\rho, \kappa, \psi, \vartheta^v \models^v \mathsf{unblind}\ E^l\ \mathsf{as}\ [E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}, \cdots, x_k]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$

iff $\rho, \vartheta^v \models^v E^l \wedge \wedge_{i=0}^{j} \rho, \vartheta^v \models^v E_i^{l_i} \wedge$
$(\forall [V_1, \cdots, V_k]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}'] \in \vartheta^v(l) : \wedge_{i=0}^{j} V_i \in \vartheta^v(l_i) \Rightarrow$
$\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi, \vartheta^v \models^v P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$
$\wedge$
$(\forall \{\![V_1, \cdots, V_{k'}]\!\}^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}']\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \vartheta^v(l) :$
$j = 0 \wedge k = 1 \wedge V_0 \in \vartheta^v(l_0) \Rightarrow$
$\{\![V_1, \cdots, V_{k'}]\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \rho(\lfloor x_1 \rfloor) \wedge$
$\rho, \kappa, \psi, \vartheta^v \models^v P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$

Table 6.4: Step 1 - Verbose formulation of the analysis.

6.3 Step 2 - From Infinite to Finite

The analysis is specified over an infinite set of values, $\mathcal{V}$, that contain infinitely many terms built from encryption and blinding. However, as described above, we can encode the infinite sets of terms into tree grammars using a finite number of rules. To prepare the reader for this transformation, we will first present a short introduction to some theory of tree grammars and how this is specifically used in the analysis, before we present the finite analysis.

The following presentation of tree grammars is based on the introduction to the subject in [9] and [13].

6.3.1 Tree Grammars

A *tree grammar* G is a finite structure (N, Σ, R, S), where N is the set of non-terminals, Σ is called a signature, R is a finite mapping of rules and $S \in N$ is a start symbol.

The signature Σ is a finite set of function symbols each associated with a number, its arity; eg. the signature $\{f_i, g_j, \ldots, h_k\}$ contains the function symbols f through h, where f has arity i, g has j, etc.

The terms are build by applying function symbols to other terms. Given a signature Σ and a set of terms X, we define the inductively smallest set of terms $T(\Sigma, X)$ as

$$T(\Sigma, X) = X \cup \{f(t_1, \ldots, t_k) \mid f_k \in \Sigma \wedge \wedge_{i=1}^{k} t_i \in T(\Sigma, X)\}$$

The function symbols with arity 0 are called constants, and if f is a constant the element $f()$ is often just written as f. A set of terms generated from an empty set of terms $T(\Sigma, \emptyset)$, that is a set generated only using constants, is called the set of *ground terms*. These sets can be regarded as a formal languages over terms, and as terms are often called trees, these languages are called *tree languages*.

Generally the rule mapping R maps terms over non-terminals into terms over non-terminals. However, as we will only work with *normalised regular tree grammars*, we require that R is a mapping from non-terminals into terms created by applying function symbols purely to non-terminals. This means that R has the functionality $R : N \to \wp(B(\Sigma, N))$ where the set B is as defined below.

$$B(\Sigma, X) = \{f(A_1, \ldots, A_k) \mid f_k \in \Sigma \wedge \wedge_{i=1}^{k} A_i \in X\}$$

Note that $B(\Sigma, X)$ is a subset of $T(\Sigma, X)$ and that the set X is not included in $B(\Sigma, X)$. If an element u is in $R(A)$, the pair (A, u) is called a rule in the grammar, and we write that A is the *head* of the rule and u is the *body* of the rule.

Given a (normalised regular) tree grammar $G = (N, \Sigma, R, S)$ one can generate a set of ground terms starting from a non-terminal A for which there is a rule in the grammar (A, u). This set, denoted $L(G, A)$, is defined as the smallest set satisfying

$$L(G, A) = \{f(t_1, \ldots, t_k) \mid f(A_1, \ldots, A_k) \in R(A) \wedge \wedge_{i=1}^{k} t_i \in L(G, A_i)\}$$

Note that also $L(G, A)$ is a subset of $T(\Sigma, \emptyset)$. Since the definition actually only needs the rule mappings, $L(G, A)$ is sometimes just written $L(R, A)$. The tree languages generated by the tree grammar G, is the set of ground terms generated by starting at the start symbol S, hence

$$L(G) \stackrel{\text{def}}{=} L(G, S)$$

As a final note about tree grammars, one may recall from tree automata theory that languages generated by normalised regular tree grammars are equivalent to recognisable tree languages, which are closed under union, intersection and complementation [13].

6.3.2 Tree Grammars for the Analysis

In the analysis, the component $\vartheta^v : Lab \to \wp(\mathcal{V})$ uses canonical values $\lfloor Val \rfloor$; ie. sets of ground terms over a signature of canonical names and encrypted and blinded values. Each of these sets constitute a tree language, and the idea is to get a finite representation of these sets by using tree grammars. This means that each set $\vartheta^v(l)$ will be represented by a unique tree grammar $(N_l, \Sigma_l, R_l, S_l)$, such that $\vartheta^v(l) = L((N_l, \Sigma_l, R_l, S_l), l)$. The components of this grammar are explained below.

Non-terminals. A specific tree grammar is used to describe all possible values a specific term E^l can evaluate to, and it is therefore natural to let the set of non-terminals be the set of labels; Lab.

Signature. Each term will be either a name, an encryption or a blinding. Therefore we will regard names as constants and encryptions and blindings as k-ary function symbols *senc*, *aenc* and *bli* representing symmetric encryptions, asymmetric encryptions and blindings respectively. In order to make the signature finite we will only use arities used in analysis of a process P. Hence for a

specific process P we have the signature

$$\begin{array}{lll} \Sigma_{\mathrm{LYSA}} & \stackrel{\mathrm{def}}{=} & \{\lfloor n \rfloor_0 \mid n \text{ is a name used in } P\} \cup \\ & & \{senc_{k+1}^{\ell,\mathcal{L}} \mid k \text{ is the arity of a symmetric encryption annotated with } \ell \text{ and } \mathcal{L} \text{ in } P\} \cup \\ & & \{aenc_{k+1}^{\ell,\mathcal{L}} \mid k \text{ is the arity of a asymmetric encryption annotated with } \ell \text{ and } \mathcal{L} \text{ in } P\} \cup \\ & & \{bli_{k+1}^{\ell,\mathcal{L}} \mid k \text{ is the arity of a blinding annotated with } \ell \text{ and } \mathcal{L} \text{ in } P\} \end{array}$$

Often the term $bli_{k+1}^{\ell,\mathcal{L}}(A_0, A_1, \ldots, A_k)$ will be written using the much more familiar $[\![A_1, \ldots, A_k]\!]_{A_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,]$ and the same style of notation will be used for symmetric and asymmetric encryptions.

Rule mappings. We want $L((Lab, \Sigma_{\mathrm{LYSA}}, R_l, S_l), l)$ to represent the set $\vartheta^v(l)$, therefore the rule mappings will overlap. So instead of storing the rule mappings of all labels in each grammar, we will use one common mapping component γ to store all label mappings.

Start symbol. As we want $L((Lab, \Sigma_{\mathrm{LYSA}}, R_l, S_l), l)$ to represent the set $\vartheta^v(l)$ the start symbol will always be l.

All grammars will be on the form $(Lab, \Sigma_{\mathrm{LYSA}}, R_l, l)$, which is often just referred to as R_l as all other components can easily be derived from this rule mapping. As all rule mappings are stored in one common component γ, all grammars will often just be referred to as γ. With these changes the components of the finite analysis are as presented in Table 6.5.

γ	:	$Lab \to B(\Sigma_{\mathrm{LYSA}}, Lab)$	Rules in the tree grammars
ρ^f	:	$\lfloor \mathcal{X} \rfloor \to \wp(Lab)$	Variable environment
κ^f	$\in$	$\wp(Lab^*)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
u	$\in$	$B(\Sigma_{\mathrm{LYSA}}, Lab)$	Bodies of tree grammar rules
ℓ	$\in$	$\mathcal{C}$	Crypto-points
l	$\in$	Lab	Labels on terms / non-terminals

Table 6.5: Components used in the finite analysis

6.3.3 The Finite Analysis

In the finite analysis we replace the evaluation of terms with the tree grammars in γ. This yields the following predicates $\rho^f, \gamma \models^f E^l$ and $\rho^f, \kappa^f, \psi, \gamma \models^f P$. The finite analysis of blinding is presented in Table 6.6 and described in detail below.

(FBli) $\rho^f, \gamma \models^f [E_1^{l_1}, \cdots, E_k^{l_k}]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\,\mathcal{L}]^l$
iff $\wedge_{i=0}^{k} \rho^f, \gamma \models^f E_i^{l_i} \wedge [l_1, \cdots, l_k]^{\ell}_{l_0}[\mathsf{dest}\,\mathcal{L}] \in \gamma(l)$

(FUBli) $\rho^f, \kappa^f, \psi, \gamma \models^f \mathsf{unblind}\ E^l\ \mathsf{as}\ [E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}^{l_{j+1}}, \cdots, x_k^{l_k}]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$
iff $\rho^f, \gamma \models^f E^l \wedge \wedge_{i=0}^{j} \rho^f, \gamma \models^f E_i^{l_i} \wedge$
$(\forall [l'_1, \cdots, l'_k]^{\ell'}_{l'_0}[\mathsf{dest}\,\mathcal{L}'] \in \gamma(l) : (\wedge_{i=0}^{j} L(\gamma, l'_i) \text{ᴇ} L(\gamma, l_i) \neq \emptyset) \Rightarrow$
$\wedge_{i=j+1}^{k} l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \psi, \gamma \models^f P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$
$\wedge$
$(\forall \{\!|l_1^{sig}|\!\}^{\ell^{sig}}_{l_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \gamma(l) : \forall [l'_1, \cdots, l'_{k'}]^{\ell'}_{l'_0}[\mathsf{dest}\,\mathcal{L}'] \in \gamma(l_1^{sig}) :$
$j = 0 \wedge k = 1 \wedge (L(\gamma, l'_0) \text{ᴇ} L(\gamma, l_0) \neq \emptyset) \Rightarrow$
$\{\!|l'_1, \cdots, l'_{k'}|\!\}^{\ell^{sig}}_{l_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}] \in \gamma(l_1) \wedge l_1 \in \rho^f(\lfloor x_1 \rfloor) \wedge$
$\rho^f, \kappa^f, \psi, \gamma \models^f P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$

Table 6.6: Step 2 - The finite analysis

The analysis of the blinding term is is rather straightforward, note however that the component ϑ^v is no longer part of the specification, but is implicitly represented by γ. In the analysis of unblinding the pattern matching is changed correspondingly. Two terms $E_1^{l_1}$ and $E_2^{l_2}$ may match if the intersection of the set of values they may evaluate to is non-empty; that is $L(\gamma, l_1) \cap L(\gamma, l_2) \neq \emptyset$. In the verbose analysis this matching is done using the membership operator ᴇ which ignores annotations, hence we now introduce an intersection operator which ignores annotations as well. This operator is defined as follows[1]:

$$S_1 \text{ᴇ} S_2 = \{V \in S_1 \cup S_2 \mid V \text{ᴇ} S_1 \wedge V \text{ᴇ} S_2\}$$

In the second part of the analysis of unblinding, we need to divide the search for signed blinded LySA terms up into two parts. First we must find all labels of

[1] This definition is a corrected version of the one presented in [9], where it was defined as $S_1 \text{ᴇ} S_2 = \{V \in S_1 \cup S_2 \mid V \in S_1 \wedge V \text{ᴇ} S_2\}$. Notice however, that the flawed version will not introduce any flaws to the analysis, as long as it is only used to check for non-emptiness.

LySA terms which are signed, then we must find all blinded sequences (of any length) of labels for terms that belong to these labels. If the analysis finds any LySA terms which satisfies the conditions, a new unblinded (but still signed) term is generated. This new term must be assigned to a fresh label, and the most natural choice for this label would be a label l_1 for the variable x_1. Therefore we extend the syntax of the unblinding process by adding labels to the variables being bound as well. This yields the following new syntax

$$P \quad ::= \quad \ldots \mid \mathsf{unblind}\ E^l\ \mathsf{as}\ [\![E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}^{l_{j+1}}, \cdots, x_k^{l_k}]\!]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\,\mathcal{L}]\,\mathsf{in}\,P$$

We also extend the analysis to ensure that $\rho^f(\lfloor x_1 \rfloor)$ holds this label.

6.4 Step 3 - Removing Polyvariance

The analysis of unblind in step 2 is *polyadic*, meaning that it works on sequences of terms of unspecified length. However predicate symbols for the Succinct Solver needs to be of a fixed arity, so the analysis must be transformed to work on families of predicates, where each family only contains members of the same arity.

The number of families are obviously infinite, but in order to analyse a specific process, only a finite number of families is needed. Recall from Chapter 5 that a process P can be typed to $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}}, \mathcal{A}_{\mathsf{AEnc}}, \mathcal{A}_{\mathsf{Bli}})$. Clearly these sets are final, and only predicates based on the last four sets are needed to analyse P.

All values are, in the finite analysis, represented in the tree grammar rules of γ. Since the bodies of these rules can contain polyadic encryptions or blindings, they have to be sorted into families of relations. To do this we introduce such a relation, β_k, such that an element in β_k is a blinded sequence of k terms. This means that the blinded term $[\![E_1^{l_1}, \cdots, E_k^{l_k}]\!]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\,\mathcal{L}]^l$ can be encoded by letting β_k contain $(l, l_0, l_1, \cdots, l_k, \ell, \mathcal{L})$. The first element l is a unique pointer into β_k, so now we can update the γ component to point to the unique pointer instead of the blinding itself.

In the implementation of the original LySA [9], polyvariance of symmetric and asymmetric encryption was removed in a similar way to the one described above, by introducing the relation σ for symmetric encryptions and α for asymmetric encryptions. These relations are also added to the analysis.

One problem remains however, the set $\mathcal{L}$. This must be encoded as well, so

we add labels to these sets in the syntax, yielding the following syntax for the blinding term and unblinding process:

$$E ::= \ldots \mid [\![E_1^{l_1}, \cdots, E_k^{l_k}]\!]^{\ell}_{E_0^{l_0}} [\mathsf{dest}\ \mathcal{L}^{l_s}]^{l}$$

$$P ::= \ldots \mid \mathsf{unblind}\ E^{l}\ \mathsf{as}\ [\![E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}^{l_{j+1}}, \cdots, x_k^{l_k}]\!]^{\ell}_{E_0^{l_0}} [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P$$

We can now describe the sets of crypto-points by using a cache δ to store each of the labels in the sets, eg. for the set $\mathcal{L}^{l_s}$ we write $\wedge_{\ell \in \mathcal{L}}\, \ell \in \delta(l_s)$. The resulting components of the finite analysis are as listed in Table 6.7.

γ^p	:	$Lab \to \wp(\lfloor \mathcal{N} \rfloor \cup Lab)$	Rules in the tree grammars
ρ^f	:	$\lfloor \mathcal{X} \rfloor \to \wp(Lab)$	Variable environment
κ_k^p	$\in$	$\wp(Lab^k)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
σ_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Symmetric encryptions
α_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Asymmetric encryptions
β_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Blindings
δ	:	$Lab \to \wp(\mathcal{C})$	Sets of crypto-point
n	$\in$	$\mathcal{N}$	Names, public and private keys
ℓ	$\in$	$\mathcal{C}$	Crypto-points
l	$\in$	Lab	Labels on terms / non-terminals

Table 6.7: Components used in the non-polyvariant analysis

The non-polyvariant analysis of blinding is as listed in Table 6.8. Notice that in the second part of the analysis of unblind, we do not know the length of the sequence which is blinded. Hence we need to iterate over all arities used for blinding or unblinding in the process, which is exactly the arities collected in the set $\mathcal{A}_{\mathsf{Bli}}$.

6.5 Step 4 - Generating ALFP

The last step is to create the generator function, $\mathcal{G}$, which takes a process as argument and returns an ALFP formula. This is an easy task, as the analysis in Table 6.8 uses analysis components over elements from a finite structured universe.

The generator function for the blinding elements is given in Table 6.10. We

(PUBli) $\rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p [E_1^{l_1}, \cdots, E_k^{l_k}]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}]^l$

iff $\wedge_{i=0}^{k} \rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p E_i^{l_i} \wedge (l, l_0, l_1, \cdots, l_k, \ell, l_s) \in \beta_k \wedge$
$l \in \gamma^p(l) \wedge \wedge_{\ell' \in \mathcal{L}}\, \ell' \in \delta(l_s)$

(PBli) $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \alpha, \beta, \delta \models^p$
$\mathsf{unblind}\ E^l\ \mathsf{as}\ [E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}^{l_{j+1}}, \cdots, x_k^{l_k}]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P$

iff $\rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p E^l \wedge \wedge_{\ell' \in \mathcal{L}}\, \ell' \in \delta(l_s) \wedge$
$\wedge_{i=0}^{j} \rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p E_i^{l_i} \wedge$
$(\forall (l', l'_0, l'_1, \cdots, l'_k, \ell', l'_s) \in \beta_k : l' \in \gamma^p(l) \Rightarrow$
$\wedge_{i=0}^{j} L(\gamma^p, l'_i) \mathbb{E} L(\gamma^p, l_i) \neq \emptyset \Rightarrow$
$\wedge_{i=j+1}^{k} l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^p, \psi, \gamma^p, \sigma, \alpha, \beta, \delta \models^p P \wedge$
$(\ell \notin \delta(l'_s) \vee \ell' \notin \delta(l_s) \Rightarrow (\ell', \ell) \in \psi))$
$\wedge$
$(\forall (l^{sig}, l_0^{sig}, l_1^{sig}, \ell^{sig}, l_s^{sig}) \in \alpha_1 : l^{sig} \in \gamma^p(l) \Rightarrow$
$\forall k' \in \mathcal{A}_{\mathsf{Bli}} : \forall (l', l'_0, l'_1, \cdots, l'_{k'}, \ell', l'_s) \in \beta_{k'} : l' \in \gamma^p(l_1^{sig}) \Rightarrow$
$j = 0 \wedge k = 1 \wedge (L(\gamma^p, l'_0) \mathbb{E} L(\gamma^p, l_0) \neq \emptyset) \Rightarrow$
$(l_1, l_0^{sig}, l'_1, \cdots, l'_{k'}, \ell^{sig}, l_s^{sig}) \in \alpha_{k'} \wedge$
$l_1 \in \rho^f(\lfloor x_1 \rfloor) \wedge l_1 \in \gamma^p(l_1) \wedge$
$\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \alpha, \beta, \delta \models^p P \wedge$
$(\ell \notin \delta(l'_s) \vee \ell' \notin \delta(l_s) \Rightarrow (\ell', \ell) \in \psi))$

Table 6.8: Step 3 - Removing polyvariance

use the convention that membership $(a, b, c) \in R$ or $a \in f(x, y, z)$ is written as the predicates $R(a, b, c)$ and $f(x, y, z, a)$ respectively, and this also means that quantifications like $\forall x \in f(y) : \cdots$ will be written as $\forall x, y : f(y, x) \Rightarrow \cdots$. The new domains for the analysis components have been given in Table 6.9.

A last note to the generator function is the predicate NEI, which is an axiomatisation of non-emptiness of language intersections. This is defined in the structure of values in the body of the grammar rules in γ^g, and recursively checks all subcomponents except the crypto-points, as it follows the style of $\mathbb{E}$ and ignores annotations. In Table 6.10 we only show the extension of NEI required for analysing blinding.

γ^g	$\in$	$\wp(Lab \times (\lfloor \mathcal{N} \rfloor \cup Lab))$	Rules in the tree grammars
ρ^g	$\in$	$\wp(\lfloor \mathcal{X} \rfloor \times Lab)$	Variable environment
κ_k^p	$\in$	$\wp(Lab^k)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
σ_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Symmetric encryptions
α_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Asymmetric encryptions
β_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Blindings
δ^g	$\in$	$\wp(Lab \times \mathcal{C})$	Sets of crypto-point
n	$\in$	$\mathcal{N}$	Names, public and private keys
ℓ	$\in$	$\mathcal{C}$	Crypto-points
l	$\in$	Lab	Labels on terms / non-terminals

Table 6.9: Components used in the ALFP formula generated

6.6 Soundness of the Implementation

That the implementation is sound means that the generator function in Table 6.10 is sound with respect to the original analysis in Table 6.2, which again means that each step in the transformation is sound with respect to the previous. That the implementation of the original LySA is sound was established in [9], so in this chapter we only need to show the soundness of the implementation of the extension with blinding.

6.6.1 Step 0 - Step 1

The first step in the transformation is to transform the succinct analysis into a verbose analysis, that this transformation is sound is shown by the following lemma.

Lemma 6.1 (Succinct analysis to verbose analysis) *The analysis* $\models^v$ *in Table 6.4 is sound with respect to the original analysis* $\models$ *in Table 6.2. That is*

$$\begin{array}{lllll} (1) & \mathsf{if} & \rho, \vartheta^v \models^v E^l & \mathsf{then} & \rho \models E^l : \vartheta^v(l) \\ (2) & \mathsf{if} & \rho, \kappa, \psi, \vartheta^v \models^v P & \mathsf{then} & \rho, \kappa, \psi \models P \end{array}$$

Proof The proof proceeds by structural induction in the definition of $\models^v$.
Part (1). Let $E = [\![E_1^{l_1}, \cdots, E_k^{l_k}]\!]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\,\mathcal{L}]$ and assume that $\rho, \vartheta^v \models^v E^l$ then

(GBli) $\mathcal{G}([\![E_1^{l_1}, \cdots, E_k^{l_k}]\!]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}\ ]^{l})$
$= \wedge_{i=0}^{k} \mathcal{G}(E_i^{l_i}) \wedge \beta_k(l, l_0, l_1, \cdots, l_k, \ell, l_s) \wedge$
$\gamma^g(l, l) \wedge \delta^g(l_s, \ell')$

(GUBli) $\mathcal{G}(\mathsf{unblind}\ E^{l}\ \mathsf{as}\ [E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}^{l_{j+1}}, \cdots, x_k^{l_k}]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\ \mathcal{L}^{l_s}\ ]\ \mathsf{in}\ P)$
$= \mathcal{G}(E^l) \wedge \wedge_{\ell' \in \mathcal{L}}\, \delta^g(l_s, \ell') \wedge \wedge_{i=0}^{j} \mathcal{G}(E_i^{l_i}) \wedge$
$(\forall\, l', l'_0, l'_1, \cdots, l'_k, \ell', l'_s : \beta_k(l', l'_0, l'_1, \cdots, l'_k, \ell', l'_s) \wedge \gamma^g(l, l') \Rightarrow$
$\wedge_{i=0}^{j} NEI(l'_i, l_i) \Rightarrow$
$\wedge_{i=j+1}^{k} \rho^g(\lfloor x_i \rfloor, l'_i)\ \wedge\ \mathcal{G}(P) \wedge$
$(\neg\delta^g(l'_s, \ell) \vee \neg\delta^g(l_s, \ell') \Rightarrow \psi(\ell', \ell)))$
$\wedge$
$(\forall\, l^{sig}, l_0^{sig}, l_1^{sig}, \ell^{sig}, l_s^{sig} : \alpha_1(l^{sig}, l_0^{sig}, l_1^{sig}, \ell^{sig}, l_s^{sig}) \wedge \gamma^g(l, l^{sig}) \Rightarrow$
$\wedge_{k' \in \mathcal{A}_{\mathsf{Bli}}}$
$(\forall\, l', l'_0, l'_1, \cdots, l'_{k'}, \ell', l'_s : \beta_{k'}(l', l'_0, l'_1, \cdots, l'_k, \ell', l'_s) \wedge$
$\gamma^g(l_1^{sig}, l') \Rightarrow$
$j = 0 \wedge k = 1 \wedge NEI(l'_0, l_0) \Rightarrow$
$\alpha_{k'}(l_1, l_0^{sig}, l'_1, \cdots, l'_{k'}, \ell^{sig}, l_s^{sig}) \wedge$
$\rho^g(\lfloor x_1 \rfloor, l_1)\ \wedge \gamma^g(l_1, l_1)\ \wedge\ \mathcal{G}(P) \wedge$
$(\neg\delta^g(l'_s, \ell) \vee \neg\delta^g(l_s, \ell') \Rightarrow \psi(\ell', \ell))))$

$\cdots$
$\wedge\wedge_{k \in \mathcal{A}_{\mathsf{Bli}}}$
$\forall l_1, l_2 : (\exists l'_1, l_{10}, \cdots, l_{1k}, \ell_1, l_{1s}, l'_2, l_{20}, \cdots, l_{2k}, \ell_2, l_{2s} :$
$\beta_k(l'_1, l_{10}, \cdots, l_{1k}, \ell_1, l_{1s}) \wedge$
$\beta_k(l'_2, l_{20}, \cdots, l_{2k}, \ell_2, l_{2s}) \wedge$
$NEI(l_{10}, l_{20}) \wedge \ldots \wedge NEI(l_{1k}, l_{2k}) \wedge$
$\gamma^g(l_1, l'_1) \wedge \gamma^g(l_2, l'_2))$
$\Rightarrow NEI(l_1, l_2)$

Table 6.10: Step 4 - Generating ALFP

from (VBli) we have the premisses:

$(i)^v \quad \wedge_{i=0}^{k} \rho, \vartheta^v \models^v E_i^{l_i}$
$(ii)^v \quad \forall V_0 \in \vartheta^v(l_0) \ldots V_k \in \vartheta^v(l_k) : [\![V_1, \cdots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}\ ] \in \vartheta^v(l)$

Hence by the induction hypothesis

$(i) \quad \wedge_{i=0}^{k} \rho \models E_i : \vartheta^v(l_i)$

and from (ABli) we can then conclude that $\rho \models [\![E_1, \cdots, E_k]\!]^{\ell}_{E_0}[\mathsf{dest}\ \mathcal{L}\ ] : \vartheta^v(l)$.

Part (2). Let $P = \mathsf{unblind}\ E^l\ \mathsf{as}\ [\![E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}, \cdots, x_k]\!]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\,\mathcal{L}\,]\,\mathsf{in}\,P$ and assume that $\rho, \kappa, \psi, \vartheta^v \models^v P$. From (VUBli) we have the four conjuncts for the hypothesis in unblinding.

$$
\begin{array}{ll}
(i)^v & \rho, \vartheta^v \models^v E^l \\
(ii)^v & \wedge_{i=0}^{j} \rho, \vartheta^v \models^v E_i^{l_i} \\
(iii)^v & \forall [\![V_1, \cdots, V_k]\!]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}'\,] \in \vartheta^v(l) : \wedge_{i=0}^{j} V_i \,\mathsf{E}\, \vartheta^v(l_i) \Rightarrow \\
 & \qquad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \ \wedge\ \rho, \kappa, \psi, \vartheta^v \models^v P \wedge \\
 & \qquad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi) \\
(iv)^v & \forall \{\!|[\![V_1, \cdots, V_{k'}]\!]^{\ell'}_{V_0}[\mathsf{dest}\,\mathcal{L}'\,]|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \vartheta^v(l) : \\
 & \qquad j = 0 \wedge k = 1 \wedge V_0 \,\mathsf{E}\, \vartheta^v(l_0) \Rightarrow \\
 & \qquad\qquad \{\!|V_1, \cdots, V_{k'}|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \rho(\lfloor x_1 \rfloor) \wedge \\
 & \qquad\qquad \rho, \kappa, \psi, \vartheta^v \models^v P \wedge \\
 & \qquad\qquad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi)
\end{array}
$$

for $(i)^v$ and $(ii)^v$ we get from the induction hypothesis that

$$
\begin{array}{ll}
(i) & \rho \models E^l : \vartheta^v(l) \\
(ii) & \wedge_{i=0}^{j} \rho \models E_i^{l_i} : \vartheta^v(l_i)
\end{array}
$$

From the definition that $\vartheta \stackrel{\text{def}}{=} \vartheta^v(l)$ for a given term E^l we can directly conclude that $\rho, \kappa, \psi \models P$, and hence that the verbose analysis is sound with respect to the succinct analysis. □

6.6.2 Step 1 - Step 2

The second step in the transformation is to describe the infinite sets in the analysis using tree languages, that this transformation is sound is shown by the following lemma.

Lemma 6.2 *The Analysis $\models^f$ in step 2 is sound with respect to the analysis $\models^v$ in step 1. That is: For all l and x let*

$$
\begin{array}{lcl}
\vartheta^v(l) & = & L(\gamma, l) \\
\rho(\lfloor x \rfloor) & = & \bigcup_{l' \in \rho^f(\lfloor x \rfloor)} \vartheta^v(l') \\
\kappa & = & \{\langle V_1, \ldots, V_k \rangle \mid \langle l_1, \ldots, l_k \rangle \in \kappa^f \ \wedge \wedge_{i=0}^{k} V_i \in \vartheta^v(l_i)\}
\end{array}
$$

Then

$$
\begin{array}{llll}
(1) & \mathsf{if} & \rho^f, \gamma \models^f E & \mathsf{then} \quad \rho, \vartheta^v \models E \\
(2) & \mathsf{if} & \rho^f, \kappa^f, \psi, \gamma \models^f P & \mathsf{then} \quad \rho, \kappa, \psi, \vartheta^v \models^v P
\end{array}
$$

Proof The proof proceeds by structural induction in the definition of $\models^f$.
Part (1). Let $E = [\![E_1^{l_1}, \cdots, E_k^{l_k}]\!]^{\ell}_{E_0^{l_0}}[\mathsf{dest}\,\mathcal{L}\,]$ and assume that $\rho^f, \gamma \models^f E^l$. From (FBli) we have the premisses:

$$\begin{array}{ll}(i)^f & \wedge_{i=0}^{k}\,\rho^f, \gamma \models^f E_i^{l_i} \\ (ii)^f & [\![l_1, \cdots, l_k]\!]^{\ell}_{l_0}[\mathsf{dest}\,\mathcal{L}\,] \in \gamma(l)\end{array}$$

Then from the induction hypothesis we get

$$(i)^v \quad \wedge_{i=0}^{k}\,\rho, \vartheta^v \models^v E_i^{l_i}$$

Using the definition of $L(\gamma, l)$ and $(ii)^f$ we have that

$$\{[\![V_1, \ldots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \mid \wedge_{i=0}^{k}\, V_i \in L(\gamma, l_i)\} \subseteq L(\gamma, l)$$

Which, according to the definition of ϑ^v, is equivalent to

$$\begin{array}{ll} & \{[\![V_1, \ldots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \mid \wedge_{i=0}^{k}\, V_i \in \vartheta^v(l_i)\} \subseteq \vartheta^v(l) \\ & \Updownarrow \\ (ii)^v & \forall V_0 \in \vartheta^v(l_0) \ldots \forall V_k \in \vartheta^v(l_k) : [\![V_1, \ldots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta^v(l)\end{array}$$

Hence we can now conclude from (VBli) that $\rho, \vartheta^v \models^v E^l$.

Part (2). Let $P' = \mathsf{unblind}\ E\ \mathsf{as}\ [\![E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}, \cdots, x_k]\!]^{\ell}_{E_0^{l_0}}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P$ and assume that $\rho^f, \kappa^f, \psi, \gamma \models^f P'$. From (FUBli) we have the four conjuncts for the hypothesis in unblinding.

$$\begin{array}{ll}(i)^f & \rho^f, \gamma \models^f E^l \\ (ii)^f & \wedge_{i=0}^{j}\,\rho^f, \gamma \models^f E_i^{l_i} \\ (iii)^f & \forall [\![l'_1, \cdots, l'_k]\!]^{\ell'}_{l'_0}[\mathsf{dest}\,\mathcal{L}'\,] \in \gamma(l) : (\wedge_{i=0}^{j}\, L(\gamma, l'_i) \,\mathbb{\in}\, L(\gamma, l_i) \neq \emptyset) \Rightarrow \\ & \qquad \wedge_{i=j+1}^{k}\, l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \psi, \gamma \models^f P \wedge \\ & \qquad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi) \\ (iv)^f & \forall \{\!|l_1^{sig}|\!\}^{\ell^{sig}}_{l_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \gamma(l) : \forall [\![l'_1, \cdots, l'_{k'}]\!]^{\ell'}_{l'_0}[\mathsf{dest}\,\mathcal{L}'\,] \in \gamma(l_1^{sig}) : \\ & \qquad j = 0 \wedge k = 1 \wedge (L(\gamma, l'_0) \,\mathbb{\in}\, L(\gamma, l_0) \neq \emptyset) \Rightarrow \\ & \qquad\qquad \{\!|l'_1, \cdots, l'_{k'}|\!\}^{\ell^{sig}}_{l_0^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \gamma(l_1) \wedge \\ & \qquad\qquad l_1 \in \rho^f(\lfloor x_1 \rfloor) \wedge \rho^f, \kappa^f, \psi, \gamma \models^f P \wedge \\ & \qquad\qquad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi)\end{array}$$

From $(i)^f$ and $(ii)^f$ the induction hypothesis gives that

$$\begin{array}{ll}(i)^v & \rho, \vartheta^v \models^v E^l \\ (ii)^v & \wedge_{i=0}^{j}\,\rho, \vartheta^v \models^v E_i^{l_i}\end{array}$$

and the definition of $\mathbb{B}$, $L(\gamma, l)$ and ϑ^v for $(iii)^f$ and $(iv)^f$ gives us directly that

$$
\begin{array}{lll}
(iii)^v & \forall [\![V_1, \cdots, V_k]\!]^{\ell'}_{V_0} [\mathsf{dest}\, \mathcal{L}'\,] \in \vartheta^v(l) : \wedge_{i=0}^{j} V_i \,\mathsf{E}\, \vartheta^v(l_i) \Rightarrow \\
& \qquad \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi, \vartheta^v \models^v P \wedge \\
& \qquad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi) \\
(iv)^v & \forall \{\!| [\![V_1, \cdots, V_{k'}]\!]^{\ell'}_{V_0} [\mathsf{dest}\, \mathcal{L}'\,] |\!\}^{\ell^{sig}}_{V_0^{sig}} [\mathsf{dest}\, \mathcal{L}^{sig}\,] \in \vartheta^v(l) : \\
& \qquad j = 0 \wedge k = 1 \wedge V_0 \,\mathsf{E}\, \vartheta^v(l_0) \Rightarrow \\
& \qquad\qquad \{\!| V_1, \cdots, V_{k'} |\!\}^{\ell^{sig}}_{V_0^{sig}} [\mathsf{dest}\, \mathcal{L}^{sig}\,] \in \rho(\lfloor x_1 \rfloor) \wedge \\
& \qquad\qquad \rho, \kappa, \psi, \vartheta^v \models^v P \wedge \\
& \qquad\qquad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi)
\end{array}
$$

and we can conclude $\rho, \kappa, \psi, \vartheta^v \models^v P'$ and hence soundness of the analysis $\models^f$ with respect to the analysis $\models^v$. □

6.6.3 Step 2 - Step 3

The third step in the transformation is to remove polyvariance in the analysis, that this transformation is sound is shown by the following lemma.

Lemma 6.3 *The analysis $\models^p$ in step 3 is sound with respect to the analysis $\models^f$ in step 2. For all l let*

$$
\begin{array}{lll}
\gamma(l) & = & \{\lfloor n \rfloor \mid \lfloor n \rfloor \in \gamma^p(l)\} \cup \\
& & \{\{l_1, \cdots, l_k\}^{\ell}_{l_0} [\mathsf{dest}\, \mathcal{L}\,] \mid l' \in \gamma^p(l) \wedge \\
& & \qquad (l', l_0, \cdots, l_k, \ell, l_s) \in \sigma_k \wedge \mathcal{L} = \delta(l_s)\} \cup \\
& & \{\{\!| l_1, \cdots, l_k |\!\}^{\ell}_{l_0} [\mathsf{dest}\, \mathcal{L}\,] \mid l' \in \gamma^p(l) \wedge \\
& & \qquad (l', l_0, \cdots, l_k, \ell, l_s) \in \alpha_k \wedge \mathcal{L} = \delta(l_s)\} \cup \\
& & \{[\![l_1, \cdots, l_k]\!]^{\ell}_{l_0} [\mathsf{dest}\, \mathcal{L}\,] \mid l' \in \gamma^p(l) \wedge \\
& & \qquad (l', l_0, \cdots, l_k, \ell, l_s) \in \beta_k \wedge \mathcal{L} = \delta(l_s)\} \\
\kappa^f & = & \bigcup_{i \in \mathbb{N}} \kappa^p_i
\end{array}
$$

Then

$$
\begin{array}{llllll}
(1) & \mathsf{if} & \rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p E & \mathsf{then} & \rho^f, \gamma \models^f E \\
(2) & \mathsf{if} & \rho^f, \kappa^p, \psi, \gamma^p, \sigma, \alpha, \beta, \delta \models^p P & \mathsf{then} & \rho^f, \kappa^f, \psi, \gamma \models^f P
\end{array}
$$

Proof The proof proceeds by structural induction in the definition of $\models^p$.
Part (1). Let $E = [\![E^{l_1}_1, \cdots, E^{l_k}_k]\!]^{\ell}_{E^{l_0}_0} [\mathsf{dest}\, \mathcal{L}^{l_s}\,]$ and assume $\rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p$

E^l. From (PBli) we have

$$\begin{array}{ll}(i)^p & \wedge_{i=0}^{k}\,\rho^f,\gamma^p,\sigma,\alpha,\beta,\delta \models^p E_i^{l_i}\\ (ii)^p & (l,l_0,l_1,\cdots,l_k,\ell,l_s)\in\beta_k\\ (iii)^p & l\in\gamma^p(l)\\ (iv)^p & \wedge_{\ell'\in\mathcal{L}}\,\ell'\in\delta(l_s)\end{array}$$

By the induction hypothesis we can conclude from (i)^p that

$$(i)^f \quad \wedge_{i=0}^{k}\,\rho^f,\gamma \models^f E_i^{l_i}$$

From the definition of γ in the lemma 6.3 we get from (ii)^p, (iii)^p and (iv)^p that

$$(ii)^f \quad [\![l_1,\cdots,l_k]\!]_{l_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,]\in\gamma(l)$$

From (FBli) we can conclude that $\rho^f,\gamma\models^f E$.

Part (2). Let $P'=\mathsf{unblind}\ E^l\ \mathsf{as}\ [\![E_1^{l_1},\cdots,E_j^{l_j};\,x_{j+1}^{l_{j+1}},\cdots,x_k^{l_k}]\!]_{E_0^{l_0}}^{\ell}[\mathsf{orig}\,\mathcal{L}^{l_s}\,]\ \mathsf{in}\ P$ and assume that $\rho^f,\kappa^p,\psi,\gamma^p,\sigma,\alpha,\beta,\delta\models^p P'$. By the definition of $\models^p$ we have the five conjuncts for the hypothesis in unblinding.

$$\begin{array}{ll}(i)^p & \rho^f,\gamma^p,\sigma,\alpha,\beta,\delta\models^p E^l\\ (ii)^p & \wedge_{\ell'\in\mathcal{L}}\,\ell'\in\delta(l_s)\\ (iii)^p & \wedge_{i=0}^{j}\,\rho^f,\gamma^p,\sigma,\alpha,\beta,\delta\models^p E_i^{l_i}\\ (iv)^p & \forall(l',l'_0,l'_1,\cdots,l'_k,\ell',l'_s)\in\beta_k : l'\in\gamma^p(l)\Rightarrow\\ & \quad \wedge_{i=0}^{j}\,L(\gamma^p,l'_i)\,\mathbb{\Pi}\,L(\gamma^p,l_i)\neq\emptyset\Rightarrow\\ & \quad\quad \wedge_{i=j+1}^{k}\,l'_i\in\rho^f(\lfloor x_i\rfloor)\ \wedge\ \rho^f,\kappa^p,\psi,\gamma^p,\sigma,\alpha,\beta,\delta\models^p P\,\wedge\\ & \quad\quad (\ell\notin\delta(l'_s)\vee\ell'\notin\delta(l_s)\Rightarrow(\ell',\ell)\in\psi)\\ (v)^p & \forall(l^{sig},l_0^{sig},l_1^{sig},\ell^{sig},l_s^{sig})\in\alpha_1 : l^{sig}\in\gamma^p(l)\Rightarrow\\ & \quad \forall k'\in\mathcal{A}_{\mathsf{Bli}} : \forall(l',l'_0,l'_1,\cdots,l'_{k'},\ell',l'_s)\in\beta_{k'} : l'\in\gamma^p(l_1^{sig})\Rightarrow\\ & \quad\quad j=0\wedge k=1\wedge(L(\gamma^p,l'_0)\,\mathbb{\Pi}\,L(\gamma^p,l_0)\neq\emptyset)\Rightarrow\\ & \quad\quad\quad (l_1,l_0^{sig},l'_1,\cdots,l'_{k'},\ell^{sig},l_s^{sig})\in\alpha_{k'}\,\wedge\\ & \quad\quad\quad l_1\in\rho^f(\lfloor x_1\rfloor)\wedge l_1\in\gamma^p(l_1)\,\wedge\\ & \quad\quad\quad \rho^f,\kappa^p,\psi,\gamma^p,\sigma,\alpha,\beta,\delta\models^p P\,\wedge\\ & \quad\quad\quad (\ell\notin\delta(l'_s)\vee\ell'\notin\delta(l_s)\Rightarrow(\ell',\ell)\in\psi)\end{array}$$

for $(i)^p$ and $(iii)^p$ we have from the induction hypothesis that

$$\begin{array}{ll}(i)^f & \rho^f,\gamma\models^f E^l\\ (ii)^f & \wedge_{i=0}^{j}\,\rho^f,\gamma\models^f E_i^{l_i}\end{array}$$

With $(ii)^p$ and the definition of γ on on the precondition of the first implication in $(iv)^p$ gives.

$$\begin{array}{ll}(iii)^f & \forall[\![l'_1,\cdots,l'_k]\!]_{l'_0}^{\ell'}[\mathsf{dest}\,\mathcal{L}'\,]\in\gamma(l) : (\wedge_{i=0}^{j}L(\gamma,l'_i)\,\mathbb{\Pi}\,L(\gamma,l_i)\neq\emptyset)\Rightarrow\\ & \qquad\qquad \wedge_{i=j+1}^{k}\,l'_i\in\rho(\lfloor x_i\rfloor)\wedge\\ & \qquad\qquad (\ell\notin\mathcal{L}'\vee\ell'\notin\mathcal{L}\Rightarrow(\ell',\ell)\in\psi)\wedge\\ & \qquad\qquad \rho,\kappa,\psi,\gamma\models^f P\end{array}$$

Using the same approach on the precondition of the first implication in $(v)^p$ gives us (a) $\forall \{\!|l_1^{sig}|\!\}_{l_0^{sig}}^{\ell^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \gamma(l)$. As $\mathcal{A}_{\mathsf{Bli}}$ holds all arities of blindings in the protocol, the precondition of the second implication is equivalent to (b) $\forall [\![l'_1, \cdots, l'_{k'}]\!]_{l'_0}^{\ell'}[\mathsf{dest}\,\mathcal{L}'\,] \in \gamma(l_1^{sig})$.

Now using the definition of γ on $l_1 \in \gamma(l_1)$ and $(l_1, l_0^{sig}, l'_1, \cdots, l'_{k'}, \ell^{sig}, l_s^{sig}) \in \alpha_{k'}$ gives us (c) $\{\!|l'_1, \cdots, l'_{k'}|\!\}_{l_0^{sig}}^{\ell^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \gamma(l_1)$. Hence from (a),(b) and (c) we have

$$
\begin{array}{ll}
(iv)^f & \forall \{\!|l_1^{sig}|\!\}_{l_0^{sig}}^{\ell^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \gamma(l) : \forall [\![l'_1, \cdots, l'_{k'}]\!]_{l'_0}^{\ell'}[\mathsf{dest}\,\mathcal{L}'\,] \in \gamma(l_1^{sig}) : \\
& \quad j = 0 \wedge k = 1 \wedge (L(\gamma, l'_0) \,\mathbb{R}\, L(\gamma, l_0) \neq \emptyset) \Rightarrow \\
& \quad\quad \{\!|l'_1, \cdots, l'_{k'}|\!\}_{l_0^{sig}}^{\ell^{sig}}[\mathsf{dest}\,\mathcal{L}^{sig}\,] \in \gamma(l_1) \wedge \\
& \quad\quad l_1 \in \rho^f(\lfloor x_1 \rfloor) \wedge \rho^f, \kappa^f, \psi, \gamma \models^f P \wedge \\
& \quad\quad (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))
\end{array}
$$

and from $(i)^f$,$(ii)^f$,$(iii)^f$ and $(iv)^f$ we can conclude that $\rho^f, \kappa^f, \psi, \gamma \models^f P'$ and that the analysis $\models^p$ is sound with respect to the analysis of $\models^f$. □

6.6.4 Step 3 - Step 4

The last step in the transformation is to transform the non-polyvariant analysis into a generator function for ALFP. The generator function only uses predicates, but as three of the components in the analysis $\models^p$ are functions, a function

$$fun(R)(a) = \{b \mid R(a, b)\}$$

is needed to map between the two domains. We can now show that this last step of the transformation is sound by the following lemma.

Lemma 6.4 *The solution to the formula generated by $\mathcal{G}$ in step 4 is sound with respect to the analysis $\models^p$ in step 3. Let*

$$
\begin{array}{lcl}
\rho^f & = & fun(\rho^g) \\
\gamma^p & = & fun(\gamma^g) \\
\delta & = & fun(\delta^g)
\end{array}
$$

then

(1) *If $(\rho^g, \gamma^g, \sigma, \alpha, \beta, \delta^g)$ satisfies $\mathcal{G}(E)$ and NEI then $\rho^f, \gamma^p, \sigma, \alpha, \beta, \delta \models^p E$*
(2) *If $(\rho^g, \kappa^p, \psi, \gamma^g, \sigma, \alpha, \beta, \delta^g)$ satisfies $\mathcal{G}(P)$ and NEI then $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \alpha, \beta, \delta \models^p P$*

Proof The proof follows directly from the definition of NEI, γ^g, ρ^g and δ^g and simple rewriting of the formula. □

6.6.5 Conclusion

The soundness of the implementation can now be summarised as follows.

Theorem 6.5 *The generation function is sound with respect to the original analysis. That is,*

(1) *If* $(\rho^g, \gamma^g, \sigma, \alpha, \beta, \delta^g)$ *satisfies* $\mathcal{G}(E)$ *and* NEI *then* $\rho, \kappa \models E : \vartheta$
(2) *If* $(\rho^g, \kappa^p, \psi, \gamma^g, \sigma, \alpha, \beta, \delta^g)$ *satisfies* $\mathcal{G}(P)$ *and* NEI *then* $\rho, \kappa, \psi \models P$

Proof The theorem follows from Lemma 6.1 to 6.4. □

Hence it follows, that the implementation of the extension with blinding is sound with respect to the analysis. Since this proof of soundness has followed the same approach as the proof for soundness of the original implementation of LySA, this shows that the entire implementation of the extended LySA is sound with respect to the analysis. A similar proof exists for the soundness of Succinct Solver [33], so we can now conclude, that a solution returned by solver to the ALFP formula is also a solution to the complete analysis of LySA with blinding as listed in Table A.5 and A.6 in the appendix. This concludes the proof for soundness.

6.7 The Attacker

In Chapter 5, the attacker was specified as a formula $\mathcal{F}^{\mathsf{DY}}$, which described its capabilities. The idea was that the analysis components should satisfy both the complete analysis specified in Table A.5 and A.6 in the appendix and the the formula for the attacker $\mathcal{F}^{\mathsf{DY}}$ as defined in Table 5.1.

Now in order to implement this, we could transform the formula $\mathcal{F}^{\mathsf{DY}}$ into ALFP using the same strategy as for the analysis, but this would be very cumbersome and would require a lot of proofs. Instead we will use the hardest attacker P_{Hard} as defined in Theorem 5.2, for which we already proved that $\rho, \kappa, \psi \models P_{\mathsf{Hard}}$ is equivalent to the formula $\mathcal{F}^{\mathsf{DY}}$ given they both are of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}}, \mathcal{A}^+_{\mathsf{AEnc}}, \mathcal{A}^+_{\mathsf{Bli}})$.

We can now find an ALFP formula equivalent to $\mathcal{F}^{\mathsf{DY}}$, by applying the generator function $\mathcal{G}$ to P_{Hard}. However, as $\mathcal{G}$ is defined on labelled LySA, we must first convert P_{Hard} into labelled LySA by adding labels to terms and crypto-sets. We do this by using two unique labels, $l_\bullet$ and $l_{\mathcal{C}}$, the first for variables and names and the second for the sets of crypto-points. This results in the following process

$$R_{\mathsf{Hard}} \stackrel{\text{def}}{=} !(R_1 \,||_{k\in\mathcal{A}_\kappa} R_2^k \,||_{k\in\mathcal{A}^+_{\mathsf{Enc}}} R_3^k \,||_{k\in\mathcal{A}^+_{\mathsf{AEnc}}} R_4^k \,||_{k\in\mathcal{A}^+_{\mathsf{Bli}}} R_5^k \,||_{k\in\mathcal{A}^+_{\mathsf{Enc}}} R_6^k \,|\, ||_{k\in\mathcal{A}^+_{\mathsf{AEnc}}} R_7^k \,||_{k\in\mathcal{A}^+_{\mathsf{Bli}}} R_8^k \,||\, R_9 \,||_{k\in\mathcal{A}_\kappa} R_{10}^k)$$

where

$$\begin{aligned}
R_1 &= \langle n^{l_\bullet}_\bullet\rangle.\langle m^{+l_\bullet}_\bullet\rangle.\langle m^{-l_\bullet}_\bullet\rangle.0 \;||_{n\in\mathcal{N}_{\mathsf{f}}}\; \langle n^{l_\bullet}\rangle.0\\
R_2^k &= (;\, z_1,\cdots,z_k).0\\
R_3^k &= (;\, z_0).\cdots(;\, z_k).\langle\{z_1,\cdots,z_k\}^{\ell_\bullet}_{z_0}[\mathsf{dest}\,\mathcal{C}^{l_{\mathcal{C}}}\,]^{l_\bullet}\rangle.0\\
R_4^k &= (;\, z_0).\cdots(;\, z_k).\langle\{\!|z_1,\cdots,z_k|\!\}^{\ell_\bullet}_{z_0}[\mathsf{dest}\,\mathcal{C}^{l_{\mathcal{C}}}\,]^{l_\bullet}\rangle.0\\
R_5^k &= (;\, z_0).\cdots(;\, z_k).\langle[\![z_1,\cdots,z_k]\!]^{\ell_\bullet}_{z_0}[\mathsf{dest}\,\mathcal{C}^{l_{\mathcal{C}}}\,]^{l_\bullet}\rangle.0\\
R_6^k &= (;\, z).(;\, z_0).\mathsf{decrypt}\ z^{l_\bullet}\ \mathsf{as}\ \{;\, z_1,\cdots,z_k\}^{\ell_\bullet}_{z_0}[\mathsf{orig}\,\mathcal{C}^{l_{\mathcal{C}}}\,]\ \mathsf{in}\ 0\\
R_7^k &= (;\, z).(;\, z_0).\mathsf{decrypt}\ z^{l_\bullet}\ \mathsf{as}\ \{\!|;\, z_1,\cdots,z_k|\!\}^{\ell_\bullet}_{z_0}[\mathsf{orig}\,\mathcal{C}^{l_{\mathcal{C}}}\,]\ \mathsf{in}\ 0\\
R_8^k &= (;\, z).(;\, z_0).\mathsf{unblind}\ z^{l_\bullet}\ \mathsf{as}\ [\![;\, z_1^{l_\bullet},\cdots,z_k^{l_\bullet}]\!]^{\ell_\bullet}_{z_0}[\mathsf{orig}\,\mathcal{C}^{l_{\mathcal{C}}}\,]\ \mathsf{in}\ 0\\
R_9 &= (;\, z).(;\, z_0).\mathsf{unblind}\ z^{l_\bullet}\ \mathsf{as}\ [\![;\, z_1^{l_\bullet}]\!]^{\ell_\bullet}_{z_0^{l_\bullet}}[\mathsf{orig}\,\mathcal{C}^{l_{\mathcal{C}}}\,]\ \mathsf{in}\ 0\\
R_{10}^k &= (;\, z_1).\cdots(;\, z_k).\langle z_1^{l_\bullet},\cdots,z_k^{l_\bullet}\rangle.0
\end{aligned}$$

Notice that also the variables being bound in the unblinding process needs labels, as defined in Section 6.3. The reason that only two labels are needed, is that the knowledge of the attacker does not need to be duplicated. This is obviously also logically equivalent to a unique labelling, as all variables have the same canonical representative $z_\bullet$.

6.8 The Extended LySATool

The analysis is implemented in New Jersey ML [36] as an extension of the original LySATool. The original LySATool was an implementation of the original analysis of LySA, and the new version of the LySATool has the exact same functionalities as the original one had; ie. it takes an annotated LySA process P as

input and returns an ALFP formula which is a conjunction of $\mathcal{G}(P)$, $\mathcal{G}(R_{hard})$ and the formula for NEI as output. The resulting ALFP formula can then be solved using the Succinct Solver.

In the actual implementation, the Succinct Solver is automatically used to compute the smallest solution that satisfies the formula. The result is then parsed and presented in a readable manner; ie. in a HTML document where each label l is hyperlinked to its corresponding rule in the tree grammar $\gamma(l)$. Additionally the implementation outputs a pretty printing LaTeXversion of the protocol P, which can easily be included in a document.

The implementation of the extension with blinding has followed the original implementation of the LySATool closely, and we have therefore been able to restrict the changes to a minimum by reusing existing functions. The changes from the implementation of the LySATool into the extended LySATool are described in detail in Appendix A.3.

Chapter 7

Analysing Protocols

In the previous chapters the distinct elements in our framework has been formalised with background in the LySA-calculus. In the following we will describe the analysis result using the LySATool [26] for the FOO92 voting protocol [18], as well as additional protocols in the FOO92 family; the Sensus voting protocol [14] and the E-Vox voting protocol [20].

For the three protocols we shall discuss the properties of interest in turn. In Section 1.3 we identified five properties a voting protocol should satisfy: Verifiability, Accuracy, Democracy, Privacy and Fairness. The privacy property is special, and we have not been able to use the analysis to validate this property, however privacy is discussed in Chapter 8.

The three protocols have the same kinds of principals; voters, administrator and counter, and the E-Vox protocol has an anonymiser as well. The FOO92 protocol is designed to run with only one administrator and one counter, while the E-Vox and Sensus protocols can also be used in a setting with multiple administrators.

The analysis provides means for analysing an arbitrarily large number rather than than a fixed number of principals. Recall from Section 4.2 that the analysis uses equivalence classes for dealing with infinite sets, and thus an infinite number of instances of one principal can be analysed by grouping these into one

equivalence group. Hence we have grouped the instances of each principal of the protocols into two groups, one that acts correct and one that is malicious, where both of these groups represent an infinite number of instances of that principal.

For readability in the LySA specifications in this chapter we have only one instance of each of the principals in the protocol; voter, administrator, counter and anonymiser. But the analysis has been run using an arbitrary large number of voters and administrators. We have not used multiple counters, since the hierarchical structure of the protocols dictates that multiple counters merely implies multiple instances of votes, each with one counter.

In our scenarios, the attacker is also an eligible voter. If the security properties are satisfied in this scenario they are obviously also satisfied if the attacker is not allowed to vote.

7.1 Assumptions

In order to analyse the protocols, we have to make some specific assumptions. These assumptions are described in or can be derived from the original protocol descriptions [18, 14, 20]:

- (Ass1) Bit-committed votes are unique, this can be seen as part of the original perfect cryptography assumption,
- (Ass2) The administrator only signs one vote for each eligible voter,
- (Ass3) The counter C is a trusted party, ie. if the counter receives a vote then it is also counted correctly in the final tally,
- (Ass4) The counter must have received all votes in the voting phase before commencing the publishing phase,
- (Ass5) The vote is only accepted if the number of votes counted by the counter equals the number of votes signed by the administrator, and
- (Ass6) The vote is only accepted if the counter in the opening phase receives all the commitment keys for the votes published.

Together with the analysis result, the assumptions are used for validating the security properties of the voting protocols.

7.2 The FOO92 Voting Protocol

The annotations in the LySA specification of the FOO92 protocol from Chapter 3 was unspecified. The security properties of interest only need specifications of the sets $\mathcal{L}_{a_1}$ and $\mathcal{L}_{c_1}$ of crypto-points, whereas the remaining are set to the entire set of crypto-points $\mathcal{C}$. The resulting LySA specification of the protocol is summarised in Table 7.1.

$$
\begin{array}{lll}
 & (\nu_{\pm}\, K_V)\,(\nu_{\pm}\, K_A) & \\
 & (\quad (\nu\, v)\,(\nu\, r)\,(\nu\, b) & /* \text{ Voter } */ \\
1. & \quad \langle V, A, V, \{\!|[\{v\}_r^{v_1}[\mathsf{dest}\,\mathcal{C}\,]]_b^{v_2}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_V^-}^{v_3}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & \\
2'. & \quad (A, V;\, x_1). & \\
2''. & \quad \mathsf{decrypt}\ x_1\ \mathsf{as}\ \{|;\, x_2|\!\}_{K_A^+}^{v_4}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} & \\
2'''. & \quad \mathsf{unblind}\ x_2\ \mathsf{as}\ [;\, x_3]_b^{v_5}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} & \\
3. & \quad \langle D, C, \{|x_3|\!\}_{K_A^-}^{v_6}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & \\
4'. & \quad (C, D, \{|x_3|\!\}_{K_A^-}^{v_7}[\mathsf{dest}\,\mathcal{C}\,];\, x_4). & \\
5. & \quad \langle D, C, x_4, r\rangle.0 & \\
 & & \\
1'. & |\quad (V, A, V;\, y_1). & /* \text{ Administrator } */ \\
1''. & \quad \mathsf{decrypt}\ y_1\ \mathsf{as}\ \{|;\, y_2|\!\}_{K_V^+}^{a_1}[\mathsf{orig}\,\mathcal{L}_{a_1}\,]\ \mathsf{in} & \\
2. & \quad \langle A, V, \{|y_2|\!\}_{K_A^-}^{a_2}[\mathsf{dest}\,\mathcal{C}\,]\rangle.0 & \\
 & & \\
 & |\quad (\nu\, l) & /* \text{ Counter } */ \\
3'. & \quad (D, C;\, z_1). & \\
3''. & \quad \mathsf{decrypt}\ z_1\ \mathsf{as}\ \{|;\, z_2|\!\}_{K_A^+}^{c_1}[\mathsf{orig}\,\mathcal{L}_{c_1}\,]\ \mathsf{in} & \\
4. & \quad \langle C, D, \{|z_2|\!\}_{K_A^-}^{c_2}[\mathsf{dest}\,\mathcal{C}\,], l\rangle. & \\
5'. & \quad (D, C, l;\, z_3). & \\
5''. & \quad \mathsf{decrypt}\ z_2\ \mathsf{as}\ \{;\, z_4\}_{z_3}^{c_3}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in}\ 0 & \\
 & & \\
 & |\quad \langle K_V^+, K_A^+\rangle.0 & /* \text{ Knowledge of the attacker } */ \\
 &) &
\end{array}
$$

Table 7.1: FOO92 in LySA-calculus

For each property we shall write the assertions that describes the property, that is we shall specify the sets $\mathcal{L}_{a_1}$ and $\mathcal{L}_{c_1}$ of crypto-points left unspecified in Table 7.1.

Verifiability. A system is verifiable if the voters independently can verify that their vote has been counted correctly. A voter can be sure of this, when he is certain that the counter has received the committed vote, due to the assumptions

(Ass3) and (Ass6), namely that the counter is trusted and that the vote will be dismissed if not all commitment keys for the votes published are received. This means that the verifiability property concerns authentication of the list published by the counter. The input (message 4') in the LySA specification must originate from the counter but in LySA we cannot add annotations to plaintext messages. We can however encode this assertion by symmetric encryption of the message from the counter with a key K, known also by the attacker, thereby not restricting the analysis. This addition to the LySA specification of the protocol is done as follows:

$$
\begin{array}{lll}
4'. & (C, D;\, x_4). & /* \text{ Voter } */ \\
4''. & \mathsf{decrypt}\ x_4\ \mathsf{as}\ \{\!|x_3|\!\}^{v_7}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,];\, x_5\}^{v_8}_{K}[\mathsf{orig}\,\mathcal{L}_{v_8}\,]\ \mathsf{in} & \\
\vdots & & \\
4. & \langle C, D, \{\!|z_2|\!\}^{c_2}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,], l\}^{c_4}_{K}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & /* \text{ Counter } */ \\
\vdots & & \\
 & \quad | \quad \langle K_V^+, K_A^+, K\rangle.0 & /* \text{ Knowledge of the attacker } */
\end{array}
$$

By taking $\mathcal{L}_{v_8} = \{c_4\}$ we require that the publication of the list must originate from the counter. This specification has been analysed together with the requirements that the sets $\mathcal{L}_{a_1}$ and $\mathcal{L}_{c_1}$ of crypt-points equal $\mathcal{C}$, the complete set of crypto-points. The analysis reports a potential attack, namely that publishing of the list can originate from the attacker: $(\ell_\bullet, v_8) \in \psi$.

The description of FOO92 [18] requires that the publication is accessible to all voters, but it does not say anything about the authentication of the list. In recent work [24] and in our interpretation of the protocol description, publication of the list is modelled by sending it on the ether, but as the analysis shows, this is not sufficient to guarantee verifiability.

This flaw has to the best of our knowledge not been reported in previous literature, but a simple amendment to the protocol can correct it. If the counter signs the list before publishing, as shown below, it turns out that the protocol is verifiable; that is, the analysis tool gives $\psi = \emptyset$. Note that the public key K_C^+ of the counter is published on the ether in order to let the attacker know it. The amendments of the protocol are:

$$
\begin{array}{lll}
4'. & (C, D;\, x_4). & /* \text{ Voter } */ \\
4''. & \mathsf{decrypt}\ x_4\ \mathsf{as}\ \{\!|x_3|\!\}^{v_7}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,];\, x_5\}^{v_8}_{K_C^+}[\mathsf{orig}\,\mathcal{L}_{v_8}\,]\ \mathsf{in} & \\
\vdots & & \\
4. & \langle C, D, \{\!|z_2|\!\}^{c_2}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,], l\}^{c_4}_{K_C^-}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & /* \text{ Counter } */ \\
\vdots & & \\
 & \quad | \quad \langle K_V^+, K_A^+, K_C^+\rangle.0 & /* \text{ Knowledge of the attacker } */
\end{array}
$$

The remaining properties to be discussed below are validated for this version of the protocol.

Accuracy. As mentioned in the introduction, accuracy of a voting protocol is obtained when (1) it is not possible for a vote to be altered, (2) invalid votes must not be counted in the final tally and (3) all validated votes must count in the final tally.

We will begin with property (2). A valid vote is a committed vote signed by the administrator, which is obtained by the voter after the unblinding at crypto-point v_6. In order for a vote to count in the final tally it has to be accepted by the counter at crypto-point c_1. Hence we shall take $\mathcal{L}_{c_1} = \{v_6\}$ whereas the other sets $\mathcal{L}_{a_1}$ and $\mathcal{L}_{v_8}$ of crypto-points are set to $\mathcal{C}$.

Analysing the protocol yields a violation to the assertions: $(a_2, c_1) \in \psi$ ie. the blinded, signed ballots can be accepted by the counter without being unblinded first. Inspecting the protocol shows that this is indeed possible because the counter accepts any new ballot which is signed by the administrator without being able to verify the content of the ballot, hence the counter is not able to distinguish between a committed vote or a blinded committed vote. This means that the attacker is able to get an arbitrarily large number n of ballots accepted by the counter in the voting phase. He can do this by blinding his ballot $n-1$ times before having it validated by the administrator and then unblinding the signed result $n-1$ times, thereby obtaining n unique values, all signed by the administrator. He will of course only be able to supply the counter with a commitment key for one of these values (the one unblinded $n-1$ times) and therefore this attack will not violate accuracy but only force the vote to be disqualified.

Clearly the existence of this denial of service attack is not very satisfactory and it can be avoided by extending the specification. As the header of an encrypted value often contains information on the type of encryption used, we extend the LySA specification by adding a header BIT to the committed vote in the first message:

1. $\langle V, A, V, \{\!|[\![BIT, \{v\}_r^{v_1}[\mathsf{dest}\,\mathcal{C}\,]\!]\!]_b^{v_2}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_V^-}^{v_3}[\mathsf{dest}\,\mathcal{C}\,]\rangle$.

And then step $3''$ only succeeds when the received messages is an unblinded message with BIT in the header:

$3''$. $\mathsf{decrypt}\ z_1\ \mathsf{as}\ \{\!|BIT;\ z_2|\!\}_{K_A^+}^{c_1}[\mathsf{orig}\,\mathcal{L}_{c_1}\,]\,\mathsf{in}$

Additionally we let the attacker know the value BIT by sending it in plaintext on the ether, as this value merely models a standard header.

Analysing the protocol in a scenario where the attacker is allowed to vote we get the violations of the assertion; $(a_2^\bullet, c_1) \in \psi$ and $(c_2, c_1) \in \psi$. The first violation $(a_2^\bullet, c_1) \in \psi$ means that the attacker may get his validated vote accepted

by the counter without unblinding it; this is equivalent to saying that the attacker can get his vote validated by the administrator without blinding it. The corresponding attack is as follows:

1. $DY \rightarrow A$: $V,\ sign_V(BIT, commit_r(v))$
2. $A \rightarrow DY$: $sign_A(BIT, commit_r(v))$
3. $DY \rightarrow C$: $sign_A(BIT, commit_r(v))$
4. $C \rightarrow DY$: $sign_C(l,\ sign_A(BIT, commit_r(v)))$
5. $DY \rightarrow C$: $l,\ r$

This attack shows that the attacker can choose not to blind his committed vote and hence be un-anonymous. However this does not violate that only valid ballots can be accepted by the counter as the attacker still needs to get his ballot validated by the administrator. Therefore we can extend the assertion $\mathcal{L}_{c_1}$ to include $a_2^\bullet$.

Turning to the violation $(c_2, c_1) \in \psi$ we observe that the assumption, that the counter must have received all votes in the voting phase before commencing the publishing phase (Ass4), contradicts that something encrypted at crypto-point c_2 can be decrypted at c_1 and hence we extend the assertion $\mathcal{L}_{c_1}$ to include c_2 as well. Now analysing the protocol with the sets $\mathcal{L}_{a_1}$ and $\mathcal{L}_{v_8}$ equal to $\mathcal{C}$ and $\mathcal{L}_{c_1} = \{v_6, a_2^\bullet, c_2\}$ we obtain an empty ψ-component which means that no invalid votes can get accepted by the counter and therefore cannot count in the final tally.

For part (1) of the accuracy property we note that, as we assume perfect cryptography, it is not possible for a vote to be altered when it has been validated. That the vote cannot be altered before validation can be observed from the possible variable bindings of x_3 in the analysis result, $\rho(x_3) = \{\{v\}_r\}$. Knowing that the analysis is an over-approximation we can be certain that only the unaltered vote can be validated and accepted by the voter, and we have that (1) is satisfied.

That all validated votes are counted, as required by part (3) of the accuracy property, relies on our assumptions and the previous parts. We know from part (2) that invalid ballots cannot be counted in the final tally. With the assumptions that the administrator only signs one ballot for each voter (Ass2), that the number of accepted votes by the counter must be the same as the number of validated votes by the administrator (Ass5) and that every accepted ballot is unique(Ass1), we can conclude that all validated votes must be counted in the final tally and thus that accuracy is satisfied.

Democracy. Democracy is obtained if (1) only eligible voters can vote and (2) they can only vote once.

Being able to vote (1) has two issues in the FOO92 protocol. Firstly, if and

only if you are an eligible voter you must be able to get your ballot validated. Secondly, only validated ballots and all validated ballots must be accepted by the authority of the tallying, but this was already established by the validation of accuracy.

That only eligible voters are able to vote is modelled by $\mathcal{L}_{a_1} = \{v_3\}$, meaning that the vote being validated by the administrator does indeed originate from the voter it is being validated for. Analysis of the protocol with this assertion (and $\mathcal{L}_{c_1}$ and $\mathcal{L}_{v_8}$ equal to $\mathcal{C}$) yields an empty ψ-component and hence the first part of democracy holds.

That eligible voters are only allowed to vote once (2) can be validated if no replay attacks can be made on the first two messages sent from the voter (messages 1 and 3). No replay attack on the first message ensures that each voter can only get one valid vote, and no replay attack on the second message ensures that validated votes are only accepted once. According to the taxonomy of replay attacks [37] there are the following categories of replay attacks:

- From which session does the replayed message come from?
  - (1) Parallel/old/current session between same pair of players as in the attacked session.
  - (2) Parallel/old session between a different pair of players.
- Who is the recipient of the replayed message?
  - (a) Intended recipient.
  - (b) Different recipient (sender of the message or third party).
- Is the message used as intended in the protocol?
  - (i) Replayed message is used with intended purpose.
  - (ii) Replayed message is used with different purpose (type-flaw attack).

A replay attack can be classified with triple from the set $\{1, 2\} \times \{a, b\} \times \{i, ii\}$. If a protocol is annotated properly the analysis finds attacks of the types $(2, *, *)$, $(*, b, *)$ and $(*, *, ii)$ where the entries $*$ can be chosen arbitrarily [27]. The only remaining replay attack is type $(1, a, i)$, which is when a message is re-sent to the intended recipient and used with intended purpose, in a parallel or new session.

A type $(1, a, i)$ replay attack on the validation from the administrator would mean that the same voter had two or more votes validated by the administrator, but this contradicts our assumptions. A type $(1, a, i)$ replay attack on the

counter would mean that the counter accepted the same vote twice, but again this contradicts our assumption that all committed ballots are unique (Ass1) and that the counter only accepts one of each (Ass3). Hence this type of replay attack is not possible according to our assumptions in Section 7.1 and as the analysis does not report any violations to our assertions we can conclude that democracy is satisfied.

Fairness. A voting protocol is fair when early results from the vote cannot be obtained; in the FOO92 specification [18] this is defined as being before the opening phase. We shall model this by eliminating the opening phase in the LySA specification and claim that if the votes are then not in the knowledge of the attacker, fairness is obtained. Running the analysis in this scenario (with all $\mathcal{L}_\ell$ equal to $\mathcal{C}$) we do indeed observe that $v \notin \rho(z_\bullet)$ thereby validating the fairness property.

It is interesting to note that the fairness property is still satisfied even when the administrator and the counter conspire. This can be validated by letting the attacker know the secret keys for both the administrator and the counter; in this way he can act on behalf of both. As already mentioned we obtain $v \notin \rho(z_\bullet)$ also in this scenario.

Summary. Table 7.2 contains the version of the FOO92 protocol that we have successfully validated using the analysis. The sets $\mathcal{L}_{a_1}$, $\mathcal{L}_{c_1}$ and $\mathcal{L}_{v_8}$ have been selected individually to capture the property of interest as described in this section.

7.3 The Sensus Voting Protocol

The Sensus voting protocol [14] builds on the FOO92 protocol and uses the same special cryptographic operations; blinding and bit-commitment. Three principals are involved in the protocol and the protocol proceeds in five phases as shown in Table 7.3 which is explained below. We use the terms and syntax from LySA for writing the cryptographic operations. The Sensus voting protocol differs from the FOO92 protocol in the way that encryptions are added to some of the messages which was not encrypted in the FOO92 protocol. This is done to keep the messages in the protocol secret, as the Sensus protocol has been implemented to work in a hostile environment, but as we shall see this does not impact on the properties specific for electronic voting.

Message 1 differs from FOO92 in the way that the voter uses the administrators

$$
\begin{array}{lll}
 & (\nu_{\pm}\ K_V)\,(\nu_{\pm}\ K_A)\,(\nu_{\pm}\ K_C)\,(\nu\, BIT) & \\
 & (\quad (\nu\, v)\,(\nu\, r)\,(\nu\, b) & /*\ \mathsf{Voter}\ */ \\
1. & \quad \langle V, A, V, \{\!|[\![BIT, \{v\}_r^{v_1}[\mathsf{dest}\,\mathcal{C}\,]]\!]_b^{v_2}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_V^-}^{v_3}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & \\
2'. & \quad (A, V;\ x_1). & \\
2''. & \quad \mathsf{decrypt}\ x_1\ \mathsf{as}\ \{|;\ x_2|\!\}_{K_A^+}^{v_4}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} & \\
2'''. & \quad \mathsf{unblind}\ x_2\ \mathsf{as}\ [;\ x_3, x_4]_b^{v_5}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} & \\
3. & \quad \langle D, C, \{|x_3, x_4|\!\}_{K_A^-}^{v_6}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & \\
4'. & \quad (C, D;\ x_5). & \\
4''. & \quad \mathsf{decrypt}\ x_5\ \mathsf{as}\ \{|\{|x_3, x_4|\!\}_{K_A^-}^{v_7}[\mathsf{dest}\,\mathcal{C}\,];\ x_6|\!\}_{K_C^+}^{v_8}[\mathsf{orig}\,\mathcal{L}_{v_8}\,]\ \mathsf{in} & \\
5. & \quad \langle D, C, x_6, r\rangle.0 & \\
 & & \\
1'. & |\quad (V, A, V;\ y_1). & /*\ \mathsf{Administrator}\ */ \\
1''. & \quad \mathsf{decrypt}\ y_1\ \mathsf{as}\ \{|;\ y_2|\!\}_{K_V^+}^{a_1}[\mathsf{orig}\,\mathcal{L}_{a_1}\,]\ \mathsf{in} & \\
2. & \quad \langle A, V, \{|y_2|\!\}_{K_A^-}^{a_2}[\mathsf{dest}\,\mathcal{C}\,]\rangle.0 & \\
 & & \\
 & |\quad (\nu\, l) & /*\ \mathsf{Counter}\ */ \\
3'. & \quad (D, C;\ z_1). & \\
3''. & \quad \mathsf{decrypt}\ z_1\ \mathsf{as}\ \{|BIT;\ z_2|\!\}_{K_A^+}^{c_1}[\mathsf{orig}\,\mathcal{L}_{c_1}\,]\ \mathsf{in} & \\
4. & \quad \langle C, D, \{|\{|BIT, z_2|\!\}_{K_A^-}^{c_2}[\mathsf{dest}\,\mathcal{C}\,], l|\!\}_{K_C^-}^{c_4}[\mathsf{dest}\,\mathcal{C}\,]\rangle. & \\
5'. & \quad (D, C, l;\ z_3). & \\
5''. & \quad \mathsf{decrypt}\ z_2\ \mathsf{as}\ \{;\ z_4\}_{z_3}^{c_3}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in}\ 0 & \\
 & & \\
 & |\quad \langle K_V^+, K_A^+, K_C^+, BIT\rangle.0 & /*\ \mathsf{Knowledge\ of\ the\ attacker}\ */ \\
 &) &
\end{array}
$$

Table 7.2: Amended FOO92 in LySA-calculus

$$
\begin{array}{llll}
1. & V & \to A & : \{|V, \{|[\![\{v\}_r]\!]_b|\!\}_{K_V^-}|\!\}_{K_A^+} \\
2. & A & \to V & : \{|\{|[\![\{v\}_r]\!]_b|\!\}_{K_A^-}|\!\}_{K_V^+} \\
3. & (V) & \to C & : \{|\{|\{v\}_r|\!\}_{K_A^-}|\!\}_{K_C^+} \\
4. & C & \to & : \{|\{v\}_r|\!\}_{K_C^-}, l \\
5. & (V) & \to C & : l, r
\end{array}
$$

Table 7.3: Protocol Narration for Sensus

private key to encrypt the message, thereby ensuring that only the administrator is able to read the message. The second and third message follows the same trend where the message is encrypted with the public key of the intended recipient of the message. When the counter publishes the list with votes in message 4, the votes are signed by the counter. The fifth message is identical to the last

message in FOO92.

7.3.1 Modelling Sensus in LySA

As for the FOO92 protocol the translation from the ordinary protocol narration into LySA proceeds in two steps. First we shall refine the specification given in Table 7.3 into an extended protocol narration which is listed in Table 7.4. The second stage translates the extended protocol narration into LySA (with

1.	$V\rightarrow$	:	$V, A, \{\!\|V, \{\!\|[\{v\}_r]_b\|\!\}_{K_V^-}\|\!\}_{K_A^+}$	
$1'$.	$\rightarrow A$	:	y_V, y_A, y_1	[check $y_A = A$]
$1''$.	A	:	decrypt y_1 as $\{\!\|y'_V, y_2\|\!\}_{K_A^-}$	
$1'''$.	A	:	decrypt y_2 as $\{\!\|y_3\|\!\}_{K_V^+}$	[check V's signature]
2.	$A\rightarrow$	:	$A, V, \{\!\|\{\!\|y_3\|\!\}_{K_A^-}\|\!\}_{K_V^+}$	
$2'$.	$\rightarrow V$	:	x_A, x_V, x_1	[check $x_V = V$]
$2''$.	V	:	decrypt x_1 as $\{\!\|x_2\|\!\}_{K_V^-}$	
$2'''$.	V	:	decrypt x_2 as $\{\!\|x_3\|\!\}_{K_A^+}$	[check $x_3 = [\{v\}_r]_b$]
$2''''$.	V	:	unblind x_2 as $[x_4]_b$	
3.	$(V)\rightarrow$	:	$D, C, \{\!\|x_4\|\!\}_{K_C^+}$	
$3'$.	$\rightarrow C$	:	z_D, z_C, z_1	[check $z_C = C$]
$3''$	C	:	decrypt z_1 as $\{\!\|z_2\|\!\}_{K_C^-}$	
$3'''$	C	:	decrypt z_2 as $\{\!\|z_3\|\!\}_{K_A^+}$	[check A's signature]
4.	$C\rightarrow$	:	$C, D, \{\!\|z_3\|\!\}_{K_C^-}, l$	
$4'$.	$\rightarrow V$	:	x_C, x_D, x_5, x_6	
$4''$.	V	:	decrypt x_5 as $\{\!\|x_7\|\!\}_{K_C^+}$	[check $x_7 = \{v\}_r$]
5.	$(V)\rightarrow$	:	D, C, x_6, r	
$5'$.	$\rightarrow C$	:	z_D, z'_C, z_l, z_r	[check $z'_C = C$]
$5''$.	C	:	decrypt z_3 as $\{z_v\}_{z_r}$	

Table 7.4: Sensus: Extended protocol narration

blinding) which can easily done by dividing the narration into 3 processes, one for each principal. The LySA specification of the protocol is given in Table 7.5.

Again we must use a small trick because rebinding of variables is not supported in the current version of LySA; the recipient of the message has to decrypt the signature and then resign the content by using the same signature.

The sets of crypto-points of interest with respect to the properties we analyse are $\mathcal{L}_{a_2}$, $\mathcal{L}_{v_{10}}$ and $\mathcal{L}_{c_2}$, whereas the remaining sets of crypto-points are $\mathcal{C}$, the

$$
\begin{array}{ll}
 & (\nu_{\pm}\ K_C)\,(\nu_{\pm}\ K_A) \\
 & (\quad (\nu\, v)\,(\nu\, r)\,(\nu\, b) \qquad\qquad /* \text{ Voter } */ \\
1. & \quad \langle V, A, \{\!|V, \{\!|[\![\{v\}_r^{v_1}[\mathsf{dest}\,\mathcal{C}\,]]\!]_b^{v_2}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_V^-}^{v_3}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_A^+}^{v_4}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \\
2'. & \quad (A, V;\ x_1). \\
2''. & \quad \mathsf{decrypt}\ x_1\ \mathsf{as}\ \{\!|;\ x_2|\!\}_{K_V^-}^{v_5}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
2'''. & \quad \mathsf{decrypt}\ x_2\ \mathsf{as}\ \{\!|;\ x_3|\!\}_{K_A^+}^{v_6}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
2''''. & \quad \mathsf{unblind}\ x_3\ \mathsf{as}\ [\![;\ x_4]\!]_b^{v_7}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
3. & \quad \langle D, C, \{\!|\{\!|x_4|\!\}_{K_A^-}^{v_8}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_C^+}^{v_9}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \\
4'. & \quad (C, D, \{\!|x_4|\!\}_{K_C^-}^{v_{10}}[\mathsf{dest}\,\mathcal{L}_{v_{10}}\,];\ x_5). \\
5. & \quad \langle D, C, x_5, r\rangle. \\
 & \\
1'. & |\quad (V, A;\ y_1). \qquad\qquad /* \text{ Administrator } */ \\
1''. & \quad \mathsf{decrypt}\ y_1\ \mathsf{as}\ \{\!|V;\ y_2|\!\}_{K_A^-}^{a_1}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
1'''. & \quad \mathsf{decrypt}\ y_2\ \mathsf{as}\ \{\!|;\ y_3|\!\}_{K_V^+}^{a_2}[\mathsf{orig}\,\mathcal{L}_{a_2}\,]\ \mathsf{in} \\
2. & \quad \langle A, V, \{\!|\{\!|y_3|\!\}_{K_A^-}^{a_3}[\mathsf{dest}\,\mathcal{C}\,]|\!\}_{K_V^+}^{a_4}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \\
 & \\
 & |\quad (\nu\, l) \qquad\qquad /* \text{ Counter } */ \\
3'. & \quad (D, C;\ z_1). \\
3''. & \quad \mathsf{decrypt}\ z_1\ \mathsf{as}\ \{\!|;\ z_2|\!\}_{K_C^-}^{c_1}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
3'''. & \quad \mathsf{decrypt}\ z_2\ \mathsf{as}\ \{\!|;\ z_3|\!\}_{K_A^+}^{c_2}[\mathsf{orig}\,\mathcal{L}_{c_2}\,]\ \mathsf{in} \\
4. & \quad \langle C, D, \{\!|z_3|\!\}_{K_C^-}^{c_3}[\mathsf{dest}\,\mathcal{C}\,], l\rangle. \\
5'. & \quad (D, C, l;\ z_r). \\
5''. & \quad \mathsf{decrypt}\ z_3\ \mathsf{as}\ \{;\ z_v\}_{z_r}^{c_4}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
 & \\
 & |\quad \langle K_C^+, K_A^+, K_V^+\rangle.0 \qquad /* \text{ Knowledge of the attacker } */ \\
 &)
\end{array}
$$

Table 7.5: Sensus in LySA-calculus

entire set of crypto-points.

7.3.2 Analysis of Sensus

Having modelled the Sensus protocol in LySA we can now proceed with the analysis. In the following we analyse each of the desired security properties in turn. We use the unspecified crypto-points in Table 7.5 for this purpose.

Verifiability. A vote is counted correctly in the final tally when the voter can verify that the committed ballot as well as the commitment key is in the hands of the counter. In order to analyse this we need to rewrite message 4 in the

LySA specification as

$$4'. \quad (C, D;\, x_5, x_6).$$
$$4''. \quad \mathsf{decrypt}\ x_5\ \mathsf{as}\ \{x_4;\ \}^{v_{10}}_{K_C^+}[\mathsf{orig}\ \mathcal{L}_{v_{10}}]\ \mathsf{in}$$

and require that the decryption in step $4''$ originates from the publication from the counter. If we let $\mathcal{L}_{v_{10}} = \{c_3\}$ an empty ψ-component implies that the voter can verify that the counter has received the ballot. From the analysis result we see that this is the case.

In the protocol specification the voter does not get any receipt when the counter has received the commitment key and therefore as in FOO92 this requires the assumption (Ass6) for the verifiability property to be satisfied.

Accuracy. Again we start with part (2) of the property, and proceed as we did for the FOO92 protocol by annotating our specification such that the counter only accepts valid ballots. Hence we set $\mathcal{L}_{c_2} = \{v_8\}$ and from the analysis result we discover the same vulnerability as in the FOO92 protocol, namely that voters can get their ballots accepted without unblinding them first. As for FOO92, the amendment consists of adding a header BIT such that the counter can check whether a ballot is unblinded or not. The first step in the LySA specification is changed to the following:

$$1. \quad \langle V, A, \{\!|V, \{\!|[\![BIT, \{v\}_r^{v_1}[\mathsf{dest}\ \mathcal{C}\,]\!]_b^{v_2}[\mathsf{dest}\ \mathcal{C}\,]\!|\}^{v_3}_{K_V^-}[\mathsf{dest}\ \mathcal{C}\,]\!|\}^{v_4}_{K_A^+}[\mathsf{dest}\ \mathcal{C}\,]\rangle$$

And the counter only accepts a ballot if it is unblinded first:

$$3'''. \quad \mathsf{decrypt}\ z_2\ \mathsf{as}\ \{\!|BIT;\ z_3|\!\}^{c_2}_{K_A^+}[\mathsf{orig}\ \mathcal{L}_{c_2}]\ \mathsf{in}$$

The analysis result yields one violation, namely $(a_3^\bullet, c_2) \in \psi$ meaning that the attacker can get his ballot validated by the administrator without blinding it ie. acting non-anonymous in the protocol. But the result also ensures that only valid ballots are accepted by the administrator and part (2) of the property is satisfied.

Turning to part (1) of the accuracy property; due to the assumption of perfect cryptography we know that the vote cannot be altered after it has been validated by the administrator. Wether the vote can be altered before the validation, can be seen from the possible variable bindings of x_4. The analysis finds $\rho(x_4) = \{\{v\}_r\}$ as an over-approximation of the values that may be bound to x_4, hence we know that the vote cannot be altered.

Part (3) of the accuracy property relies on the assumptions from the beginning of this chapter, more specific that the number of accepted ballots at the counter must be the same as the number of validated ballots by the administrator (Ass5). As we also know that accepted ballots are unique (Ass1) and from part (2) that only valid ballots are accepted, we can conclude that all validated ballots are

accepted by the counter.

Democracy. Assume that the attacker is not an eligible voter. Then the first part of democracy is satisfied if (1) the attacker is not able to get a ballot validated by the administrator; ie. $\mathcal{L}_{a_2} = \{v_3\}$, (2) the attacker cannot get anything accepted at the counter; ie. $\mathcal{L}_{c_2} = \{v_8\}$. The analysis of the protocol with these assertions yields no violations, thus we can conclude that only eligible voters can vote.

We turn to the second part of democracy; ie. that eligible voters can vote only once. From the analysis of the FOO92 protocol we mentioned that the analysis covers the eight different replay attacks from [37] except one, which due to our assumptions ((Ass1), (Ass2) and (Ass3)) is not a possible attack. Analysing the Sensus protocol with respect to replay attacks on the above annotations; namely $\mathcal{L}_{a_2} = \{v_3\}$ and $\mathcal{L}_{c_2} = \{v_8\}$, yields no violations, and we can conclude that the Sensus protocol ensures democracy.

Fairness. As for FOO92 the Sensus protocol is divided into phases and the fairness property is satisfied if no early result from the vote can be obtained. In the Sensus protocol the result of a vote is considered early if it is obtained before the opening of the votes; ie. before message 5. Hence we eliminate step 5. in the specification and analyse for confidentiality of the votes v. From the analysis result we see that the votes are held confidential, even in a scenario where the administrator and the counter conspire, exactly as in the FOO92 analysis.

7.4 The E-Vox Voting Protocol

Another interesting voting protocol is the E-Vox voting protocol [17, 20]. This is also a variant of the FOO92 protocol but it is slightly changed in that the voters are not required to have a digital signature as they do in the two other protocols mentioned here.

The E-Vox voting protocol employs four principals; there are multiple voters V, multiple administrators A, one counter C and one anonymiser N providing anonymous communication between the voters and the counter. The E-Vox voting scheme has been proposed to also have multiple administrators. The protocol narration for E-Vox, given in Table 7.6, is further explained below.

First the voter creates a fresh key K_{VA} and sends it to the administrator encrypted under the administrators public key (message 1).

1.	$V \rightarrow A$	:	$\{\!\|K_{VA}\|\!\}_{K_A^+}$
2.	$V \rightarrow A$	:	$\{V, pwd, [\{v\}_r]_b\}_{K_{VA}}$
3.	$A \rightarrow V$	:	$\{\!\|[\{v\}_r]_b\|\!\}_{K_A^-}$
4.	$V \rightarrow N$	:	$\{\!\|K_{VC}\|\!\}_{K_C^+}, \{\{\!\|\{v\}_r\|\!\}_{K_A^-}, r\}_{K_{VC}}$
5.	$N \rightarrow C$	:	$\{\!\|K_{VC}\|\!\}_{K_C^+}, \{\{\!\|\{v\}_r\|\!\}_{K_A^-}, r\}_{K_{VC}}$
6.	$C \rightarrow$	:	$\{\!\|\{v\}_r\|\!\}_{K_A^-}, r$

Table 7.6: Protocol Narration for E-Vox

As in the FOO92 and Sensus protocols the voter selects a vote, bitcommits it and blinds it. The blinded, committed ballot together with the voters name V and a password *pwd* is sent to the administrator, encrypted under the freshly created shared key K_{VA}.

The password is used to identify the voter, replacing the voters signature in the previous protocols. Upon receiving message 2, the administrator thus checks that the password is valid for the voter. If this is the case, he signs the blinded, committed vote and returns it to the voter; message 3 in the narration.

The voter verifies the administrators signature on the blinded, committed ballot and then unblinds the it. In message 4 in the protocol narration the voter sends two messages: a fresh symmetric key K_{VC} encrypted under the counters public key, and the valid ballot together with the commitment key encrypted under the key K_{VC}. The message is send to the anonymiser which in message 5 in the protocol narration just forwards message 4, thereby not letting the counter know the identity of the voter.

When the counter receives the message, he obtains the session key K_{VC} by decrypting with his private key. Then the validation of the ballot is checked and if it succeeds, the counter uncommits the ballot and counts the vote. In message 6 the counter publishes the valid ballots that has been counted together with the corresponding commitment keys.

7.4.1 Modelling E-Vox in LySA

We refine the protocol narration given in Table 7.6 into an extended protocol narration which is given in Table 7.7. Again observe that each message is extended with source and destination information.

1.	$V\rightarrow$	:	$V, A\ \{\!\|K_{VA}\|\!\}_{K_A^+}$	
$1'$.	$\rightarrow A$	:	y_V, y_A, y_1	[check $y_A = A$]
$1''$.	A	:	decrypt y_1 as $\{\!\|y_K\|\!\}_{K_A^-}$	
2.	$V\rightarrow$	:	$V, A\ \{V, pwd, [\{v\}_r]_b\}_{K_{VA}}$	
$2'$.	$\rightarrow A$	:	y'_V, y'_A, y_2	[check $y'_A = A$]
$2''$.	A	:	decrypt y_2 as $\{y''_V, y_p, y_3\}_{yK}$	[check $y''_V = V$ and $y_p = pwd$]
3.	$A\rightarrow$	:	$A, V, \{\!\|y_3\|\!\}_{K_A^-}$	
$3'$.	$\rightarrow V$	:	x_A, x_V, x_1	[check $x_V = V$]
$3''$.	V	:	decrypt x_1 as $\{\!\|x_2\|\!\}_{K_A^+}$	[check $x_2 = [\{v\}_r]_b$]
$3'''$.	V	:	unblind x_1 as $[x_3]_b$	
4.	$V\rightarrow$	:	$V, N, \{\!\|K_{VC}\|\!\}_{K_C^+}, \{x_3, r\}_{K_{VC}}$	
$4'$.	$\rightarrow N$	:	w_V, w_N, w_1, w_2	[check $w_N = N$]
5.	$N\rightarrow$	:	N, C, w_1, w_2	
$5'$.	$\rightarrow C$	:	z_N, z_C, z_1, z_2	[check $z_C = C$]
$5''$.	C	:	decrypt z_1 as $\{\!\|z_K\|\!\}_{K_C^-}$	
$5'''$.	C	:	decrypt z_2 as $\{z_3, z_r\}_{z_K}$	
$5''''$.	C	:	decrypt z_3 as $\{\!\|z_4\|\!\}_{K_A^+}$	[check A's signature]
$5'''''$.	C	:	decrypt z_4 as $\{z_5\}_{z_r}$	
6.	$C\rightarrow$	:	C, D, z_3, z_r	
$6'$.	$\rightarrow V$	:	x_C, x_D, x_4, x_5	[check $x_5 = r$]
$6''$.	V	:	decrypt x_4 as $\{\!\|x_6\|\!\}_{K_A^+}$	[check A's signature]
$6'''$.	V	:	decrypt x_6 as $\{x_v\}_r$	[check $x_v = v$]

Table 7.7: E-Vox: Extended protocol narration

The extended narration can easily be translated into LySA by dividing the narration into four processes. The LySA specification of the protocol is given in Table 7.8. Note that the trick for validating signatures

We follow the trend from FOO92 and Sensus and leave the sets of crypto-points $\mathcal{L}_{v_9}$, $\mathcal{L}_{a_2}$ and $\mathcal{L}_{c_3}$ unspecified, and the remaing are set to the entire set of crypto-points, $\mathcal{C}$.

7.4.2 Analysis of E-Vox

Analysis of the E-Vox voting protocol proceeds as for the FOO92 and Sensus protocols where the unspecified crypto-points in Table 7.8 are specified for each of the properties.

Verifiability. We keep in mind that a system is verifiable if the voters inde-

$$
\begin{array}{ll}
 & (\nu_{\pm}\ K_C)\,(\nu_{\pm}\ K_A)\,(\nu\ pwd) \\
 & (\quad (\nu\, v)\,(\nu\, r)\,(\nu\, b) \qquad\qquad /* \text{ Voter } */ \\
 & \quad (\nu\, K_{VA})\,(\nu\, K_{VC}) \\
1. & \quad \langle V, A, \{\!|K_{VA}|\!\}^{v_1}_{K_A^+}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \\
2. & \quad \langle \{V, pwd, [\{v\}^{v_2}_r[\mathsf{dest}\,\mathcal{C}\,]]^{v_3}_b[\mathsf{dest}\,\mathcal{C}\,]\}^{v_4}_{K_{VA}}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \\
3'. & \quad (A, V;\, x_1). \\
3''. & \quad \mathsf{decrypt}\ x_1\ \mathsf{as}\ \{\!|;\, x_2|\!\}^{v_5}_{K_A^+}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
3'''. & \quad \mathsf{unblind}\ x_2\ \mathsf{as}\ [;\, x_3]^{v_6}_b[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
4. & \quad \langle V, N, \{\!|K_{VC}|\!\}^{v_7}_{K_C^+}[\mathsf{dest}\,\mathcal{C}\,], \{\{\!|x_3|\!\}^{v_8}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,], r\}^{v_9}_{K_{VC}}[\mathsf{dest}\,\mathcal{L}_{v_9}\,]\rangle. \\
6'. & \quad (C, D, r;\, x_4). \\
6''. & \quad \mathsf{decrypt}\ x_4\ \mathsf{as}\ \{\!|;\, x_5|\!\}^{v_1 0}_{K_A^+}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
6'''. & \quad \mathsf{decrypt}\ x_5\ \mathsf{as}\ \{v;\,\}^{v_1 1}_r[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
 & \\
1'. & \mid \quad (V, A;\, y_1). \qquad\qquad /* \text{ Administrator } */ \\
1''. & \quad \mathsf{decrypt}\ y_1\ \mathsf{as}\ \{\!|;\, y_K|\!\}^{a_1}_{K_A^-}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
2'. & \quad (V, A;\, y_2). \\
2''. & \quad \mathsf{decrypt}\ y_2\ \mathsf{as}\ \{V, pwd;\, y_3\}^{a_2}_{y_K}[\mathsf{orig}\,\mathcal{L}_{a_2}\,]\ \mathsf{in} \\
3. & \quad \langle A, V, \{\!|y_3|\!\}^{a_3}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \\
 & \\
4'. & \mid \quad (V, N;\, w_1, w_2). \qquad\qquad /* \text{ Anonymiser } */ \\
5. & \quad \langle N, C, w_1, w_2\rangle. \\
 & \\
5'. & \mid \quad (N, C;\, z_1, z_2). \qquad\qquad /* \text{ Counter } */ \\
5''. & \quad \mathsf{decrypt}\ z_1\ \mathsf{as}\ \{\!|;\, z_K|\!\}^{c_1}_{K_C^-}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
5'''. & \quad \mathsf{decrypt}\ z_2\ \mathsf{as}\ \{;\, z_3, z_r\}^{c_2}_{z_K}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
5''''. & \quad \mathsf{decrypt}\ z_3\ \mathsf{as}\ \{\!|;\, z_4|\!\}^{c_3}_{K_A^+}[\mathsf{orig}\,\mathcal{L}_{c_3}\,]\ \mathsf{in} \\
5'''''. & \quad \mathsf{decrypt}\ z_4\ \mathsf{as}\ \{;\, z_5\}^{c_4}_{z_r}[\mathsf{orig}\,\mathcal{C}\,]\ \mathsf{in} \\
6. & \quad \langle C, D, z_r, z_3\rangle. \\
 & \\
 & \mid \quad \langle K_C^+, K_A^+\rangle.0 \qquad\qquad /* \text{ Knowledge of the attacker } */ \\
 &)
\end{array}
$$

Table 7.8: E-Vox in LySA-calculus

pendently can verify that their votes has been counted correctly, and we use the assumption that a vote is counted correctly when it is received by the counter (Ass3). Focusing on who can receive the committed ballot and the commitment key in step 4 in the LySA specification we set $\mathcal{L}_{v_9} = \{c_2\}$ such that only the counter may receive the commitment key and thereby being the only able to open the ballot.

The analysis result does not report any violations and hence only the voter and

the counter knows the commitment key. Therefore, if the voter sees a publication of this bitcommit ballot and commitment key, he can be sure that the counter has received the vote, and that it has been counted correctly. Note that the publication of the list does not need to be signed by the counter because even if the publication is done by the attacker, the voter can verify that the counter has received the ballot, and hence according to (Ass3) the vote must also count in the final tally.

Accuracy. That only valid votes are counted in the final tally is checked analogous to the FOO92 and Sensus protocols. Acceptance of a ballot by the counter is done at crypto-point c_3, and valid ballots are generated after unblinding the ballot at crypto-point v_8. We set $\mathcal{L}_{c_3} = \{v_8\}$ and from the analysis result we get no violations and therefore only valid votes are counted in the final tally which concludes the second part of the accuracy property. Note that we do not need to add the BIT header as the commitment key is sent along with the committed ballot, and hence the counter can check the content of the commitment at once.

Turning to the first and third part of the property we can similar as for FOO92 and Sensus, argue for the properties that votes cannot be altered and all votes are counted in the final tally.

Democracy. We do not allow the attacker to be a valid voter and we divide analysis of the property into the two problems; getting a ballot validated by the administrator and getting a valid ballot accepted by the counter. A ballot is validated by the administrator if it is accepted ie. the voter has a valid password pwd in the matching in the decryption at crypto-point a_2. To validate this we set $\mathcal{L}_{a_2} = \{v_4\}$ in the LySA specification and the analysis result yields no violations and hence only eligible voters can get a valid ballot. A ballot is accepted by the counter when it is valid which is checked at crypto-point c_3 and the valid ballot is obtained by the voter after unblinding the ballot at crypto-point v_8 and hence the annotation $\mathcal{L}_{c_3} = \{v_8\}$. The analysis does not yield any violations, and we can conclude that only eligible voters can vote.

That voters can vote only once is investigated by analysis of replay attacks on the issues above, namely validation of ballots and acceptance of valid ballots. As for the previous protocols, LySA covers 7 of the 8 types of replay attacks and the last is covered by our assumptions. Hence, as the analysis does not report any violations to the annotations, we can conclude that democracy is satisfied.

Fairness. The fairness property is clearly not satisfied as the counter is able to uncommit a ballot as it has been received in step $5'$ in the LySA specification and thereby get knowledge on the result as the voters are sending their ballots. If we make an additional assumption the property can be satisfied; if the counter does not uncommit or publish the ballots before all votes has been received,

fairness is satisfied. This can be seen by excluding steps $5''''$ and 6 from the LySA specification of the protocol and then analyse the protocol with respect to confidentiality on the value v.

Chapter 8

Discussion of Part I

In Chapter 7 we have validated the four security properties; verifiability, accuracy, democracy and fairness of three different voting protocols. However, as stated in Section 1.3, the design goals for voting protocols include an additional security property; namely privacy. In this chapter we shall discuss the problems concerning validation of this property, and furthermore we shall discuss the foundation of our analysis; the assumptions.

8.1 Privacy

In Section 1.3 we define privacy as satisfied if nobody can link any vote to the voter who cast it. Usually this property is divided into two parts [14]; (1) neither election authorities nor anyone else can link any vote to the voter who cast it, and (2) no voter can prove that he or she voted in a particular way.

The analysis of LySA, uses an over-approximation of all variable bindings and all messages sent on the network to ensure destination and origin authentication. This analysis gives us all values learned by the attacker; ie. all values in $\rho(z_\bullet)$, but not how he obtained this information. This means that when we analyse a voting protocol in LySA, even if the resulting $\rho(z_\bullet)$ holds both all voters and all their votes, we cannot know if the attacker can link these values together. This

is the case for both FOO92 and Sensus, as at one point the name of the voter but not the vote is sent in clear and at another point the opposite is done. Hence we cannot use LySA to validate privacy for voting protocols, unless the attacker never learns either the name of the voter or the vote, and since he will always learn both of these if he is in allegiance with the counter and the administrator, we can never use LySA to validate the privacy property.

Turning to related work, Kremer and Ryan [24] proved that the first part of privacy was satisfied for the FOO92 protocol using equivalence theory. The proof proceeds by taking two processes P_1 and P_2, where in P_1 the voter V_1 votes $vote_1$ and the voter V_2 votes $vote_2$ whereas in P_2 the voter V_1 votes $vote_2$ and V_2 votes $vote_1$, and then show that these processes are observational equivalent. This proof could probably be used to prove the first part of privacy for E-Vox and Sensus as well, however a more interesting aspect of Kremer and Ryans work is that they too failed to use an automated tool to show this property and in particular that they were not able to validate the second part of privacy at all.

The second part of the privacy property is in [14] described as being of importance to prevent vote buying and extortion. However, they proceed in [14] by claiming that this property could never be satisfied for electronic voting protocols, unless voting booths were used, as voters could let another party observe while they voted.

This is not a problem concerning the voting protocol, but more the voting scenario in our opinion, but a much more critical problem arises when trying to validate this property. In all voting protocols, the voter must have some kind of secret; eg. a blinding factor or a cryptographic key, which protects his identity or hides his vote during the vote. If he does not, then the attacker will be able to link the voter and his vote together directly from the messages sent. This secret can always be used as proof, that the voter is the one who cast a specific vote, at least this applies to all voting protocols we have analysed. Thus we believe, that this property can never be satisfied for an electronic voting protocol in presence of the Dolev-Yao attacker, but a more formal proof of this is yet to be done.

8.2 Assumptions

The actual contribution of an analysis is always dependent of the assumptions it is based on. Assumptions are critical for any analysis, yet too many or too strong assumptions can result in the result being useless.

In Section 7.1 we state that our analysis is done under the following assumptions:

- (Ass1) Bit-committed votes are unique;
- (Ass2) The administrator only signs one vote for each eligible voter;
- (Ass3) The counter C is a trusted party, ie. if the counter receives a vote then it is also counted correctly in the final tally;
- (Ass4) The counter must have received all votes in the voting phase before commencing the publishing phase;
- (Ass5) The vote is only accepted if the number of votes counted by the counter equals the number of votes signed by the administrator; and
- (Ass6) The vote is only accepted if the counter in the opening phase receives all the commitment keys for the votes published.

which are more or less derived directly from the protocol descriptions.

The first assumption (Ass1) is intuitively needed, but it is, as stated, covered by perfect cryptography.

The next two assumptions (Ass2) and (Ass3) basically say that the administrator and the counter need to be trusted. These assumptions are a weakening point of the analysis, as we cannot always trust these parties. However, in LySA all messages are sent on the ether, which means that the attacker can see all messages sent, but so can everyone else. Hence our analysis result is still true, if each principal receives all messages sent, which again means that each voter could check the result of the vote (act as a counter), and verify that the administrator signed only one vote for each eligible voter. In essence, the two assumptions (Ass2) and (Ass3) are not critical for the validation of the security properties.

The fourth assumption (Ass4) is needed because of the protocol design, as shown in the proof for first part of privacy in [24]. However, for the four security properties we validate, this assumption is not needed.

The last two assumptions (Ass5) and (Ass6) are needed for several of the properties to be satisfied. These assumptions show that any eligible voter can force the election to be disqualified. This is a result of the protocol design and should be kept in mind, when using one of these voting protocols.

Part II

LySA^{XP}

Chapter 9

Motivation for a new Calculus

In this part of the thesis we shall present a new process calculus for specifying protocols, and a corresponding control flow analysis to safely approximate the behavior of a protocol. The process calculus presented in the following is inspired by the LySA-calculus, and is therefore named LySA$^{\mathsf{XP}}$; we shall leave the interpretation of the superscript XP up to the readers imagination. We begin by describing the stimuli for this calculus and analysis.

The main objective of LySA was that it should provide an easy way of translating ordinary protocol narrations into process calculus specifications, which an analysis could be applied to. However, as discovered in the previous part of this thesis, this translation is not always trivial. It is sometimes needed to change the order of the elements in a message, and at some occasions it is even necessary to make small hacks in the LySA specification, to express the actual intention of the protocol.

Changing the order of elements in a message concerns the pattern matching in LySA, and as an example consider message 4 in the FOO92 protocol:

$$4. \quad C \rightarrow (V) \quad : \quad l,\ \{\!|\{v\}_r|\!\}_{K_A^-}$$

Upon receiving the message, the voter will pattern match on the second element of the tuple, which should equal his own vote (committed and signed). Then, if this pattern match succeeds, the first element should be bound to a variable.

Pattern matching in LySA succeeds when the first j elements in the tuple match, and the remaining $k - j$ elements are then bound to variables, so in order to specify message 4 in the LySA-calculus we need to change the order in the tuple:

$$4. \quad C \to (V) \quad : \quad \{\!|\{v\}_r|\!\}_{K_A^-}, l$$

This change to the specification seems only minor but one could easily imagine protocols with more messages and more elements, where these slight changes could stress errors in the specification.

A more annoying limitation of LySA is that it does not support rebinding of variables. This was exemplified in the first part of the thesis, where we in the specifications of FOO92, Sensus and E-Vox had to use a small hack; eg. this was done in the FOO92 specification of messages 2 and 3:

$$\begin{array}{ll} 2'. & (A, V; x_1). \\ 2''. & \mathsf{decrypt}\ x_1\ \mathsf{as}\ \{\!|; x_2|\!\}^{v_4}_{K_A^+}[\mathsf{orig}\,\mathcal{C}\,]\,\mathsf{in} \\ 2'''. & \mathsf{unblind}\ x_2\ \mathsf{as}\ [; x_3]^{v_5}_b[\mathsf{orig}\,\mathcal{C}\,]\,\mathsf{in} \\ 3. & \langle D, C, \{\!|x_3|\!\}^{v_6}_{K_A^-}[\mathsf{dest}\,\mathcal{C}\,]\rangle. \end{array}$$

The voter has to check the administrators signature, but the only way to describe this in LySA is to first decrypt the signature and then resign the message using the same signature. This means that the voter has to use the administrators private key in the specification.

A result of these limitations is that the general use of LySA requires much insight into the analysis, and that protocol specifications using LySA often becomes very hard to read. Additionally we find that the use of asymmetric encryption to model digital signatures and the lack of a hashing construct is rather annoying. This results in a strong motivation for developing a new calculus.

The remaining part of the thesis proceeds as follows: In Chapter 10 we present the syntax and semantics LySA$^{\mathsf{xp}}$. To show that this calculus accommodates the requirements considered above, we show in Chapter 11 how to model protocols in LySA$^{\mathsf{xp}}$. Then in Chapter 12, we present the analysis of LySA$^{\mathsf{xp}}$ and a proof for soundness of the analysis with respect to the semantics and additionally we shall briefly describe how to specify the attacker. At last we will discuss some aspects of the new calculus and the analysis in Chapter 13, before we present the conclusion of the thesis in Chapter 14.

Chapter 10

LySA^{XP}-calculus

The motivation for creating a new calculus was given in the previous chapter, in this chapter we shall present the design of LySA^{XP}, and proceed by adding a formal syntax and semantics of this new calculus.

The design of the LySA^{XP}-calculus is partly inspired by the LySA^{NS}-calculus presented in [10]. However, as implementation of the analysis of LySA^{NS}-calculus has proved very challenging, a novelty of LySA^{XP} is the more efficiently implementable analysis presented in Chapter 12.

10.1 Design

In Chapter 9 we discussed some limitations of LySA; the rigid pattern matching and the lack of rebinding variables. To accommodate the first limitation, we need to make a calculus, that in a flexible manner can pattern match. We do this by introducing *patterns*. Using the notion that a value is a term without variables, we require that pattern matching of a value v against a pattern p will result in a boolean answer. The resulting syntax is as follows

$$\mathsf{match}\ v\ \mathsf{as}\ p.\ P'$$

for the process of matching a value v against a pattern p and only if the matching succeeds, should the remainder P' be evaluated. The idea is that patterns should have a syntax similar to that of terms, and continuing this idea of pattern matching, these patterns must have some means to bind new variables. This means that the syntax of patterns must include a construct for binding a value to a variable x if value has been matched successfully against a pattern p:

$$p\%x$$

Which would solve the problem in LySA concerning rebinding of variables.

In the general design of LySA^{XP}, we shall make two levels of abstraction. First we have the basic elements in the calculus; terms and patterns, as described above. On top of these we build an object-level describing the principals in a protocol; the processes.

10.2 Syntax

As described above, the syntax of LySA^{XP} is divided into two levels of abstraction. In this section we shall keep this distinction of the levels in the presentation of the syntax and the semantics and describe the syntax for both of these levels.

10.2.1 Terms and Patterns

The basic building blocks used for modelling processes are terms and patterns. The terms in LySA^{XP} are somewhat similar to those in LySA, but in LySA^{XP} we distinguish between digital signatures and symmetric encryptions, and we introduce a special construct for hashing. Also a tuple construct has been added, such that concatenation of terms is done separately. The terms in LySA^{XP} are given in Table 10.1 and explained in the following.

Beside the names n and variables x, the terms of the calculus include a tuple construct $\mathsf{T}(t_1, \cdots, t_k)$ which is used to concatenate terms. Furthermore we have terms for cryptographic operations such as symmetric encryption $\mathsf{E}_{t_0}(t)$, asymmetric encryption $\mathsf{P}_{t_0}(t)$, digital signatures $\mathsf{S}_{t_0}(t)$ and of course the blinding construct $\mathsf{B}_{t_0}(t)$. It is important to notice the effect of the tuple construct, a blinding of k values would in the original LySA be specified as $[\![V_1, \cdots, V_k]\!]_{V_0}$ whereas the blinding should be encoded as $\mathsf{E}_{v_0}(\mathsf{T}(v_1, \cdots, v_k))$ in the LySA^{XP} -calculus. It might seem that the tuple construct unessentially increases the complexity of the syntax, but this construct eases the analysis, as we shall

t	::=	n	name
	|	x	variable
	|	$\mathsf{T}(t_1, \cdots, t_k)$	tuple
	|	$\mathsf{H}(t)$	hashing
	|	$\mathsf{E}_{t_0}(t)$	symmetric encryption
	|	$\mathsf{P}_{t_0}(t)$	asymmetric encryption
	|	$\mathsf{B}_{t_0}(t)$	blinding
	|	$\mathsf{S}_{t_0}(t)$	signature

Table 10.1: Terms; t

see later. Additionally, it seems obvious to let a parser handle the tuple construct, such that it becomes invisible to the user; eg. symmetric encryption of a tuple $\mathsf{T}(t_1, \cdots, t_k)$ with a key K would be written $\mathsf{E}_K(t_1, \cdots, t_k)$ instead of $\mathsf{E}_K(\mathsf{T}(t_1, \cdots, t_k))$.

In addition to the cryptographic operations from LySA, LySA$^{\text{XP}}$ also incorporates a special construct for hashing; $\mathsf{H}(t)$. Note that hashing does not need a key, as it can never be decrypted.

The other basic element in LySA$^{\text{XP}}$ is patterns. As mentioned in Chapter 9, patterns are used to match on values and possibly bind these to variables. Hence the syntax for patterns should follow the syntax for terms, except that it should incorporate a construct for binding of variables. The resulting syntax for patterns is given in Table 10.2. Beside matching of cryptographic operations the basic patterns include binding of variables as we described in the introduction; matching a value v against the pattern $p\%x$, means matching v against p and if this succeeds the variable x is bound to v. Notice the wildcard _ which matches any value, this means that matching a value v against $\_\%x$ always succeeds and binds x to v.

The syntax for pattern matching introduces two additional patterns; a *constructive pattern* and a *signature pattern*. The constructive pattern is used for keys and hashed terms, and hence this pattern does not allow wildcards or binding of values. This means that one cannot decrypt a cryptographic term without knowing the exact key, nor can one learn anything about the key of a cryptographic term. The same pattern applies to hashed values, as one must know all values used for a hashing in order to pattern match against it, and one cannot learn anything about the values used to create the hashing.

The signature pattern is used when we want to verify a signatures, ie. matching a term against the pattern $\mathsf{S}_{sp_0}(p)$. The signature pattern is either like the

p	::=	n	matches name
	|	x	matches value of variable
	|	_	matches anything
	|	$p\%x$	binds variable x when p matches
	|	$\mathsf{T}(p_1, \cdots, p_k)$	matches tuple
	|	$\mathsf{H}(cp)$	matches hashed values
	|	$\mathsf{E}_{cp}(p)$	symmetric decryption
	|	$\mathsf{P}_{cp}(p)$	asymmetric decryption
	|	$\mathsf{B}_{cp}(p)$	unblinding
	|	$\mathsf{S}_{sp}(p)$	verify signature
cp	::=	$n \mid x \mid \mathsf{T}(cp_1, \cdots, cp_k) \mid \ldots$	constructive pattern - as p but no wildcard or binding of variables
sp	::=	_ | cp	signature pattern

Table 10.2: Patterns; p

constructive pattern or just a wildcard. This means that everybody can read the signed content, but if one wants to verify the signature the exact key must be known.

Notice that in LySA digital signatures was modelled by asymmetric encryption using ones private key. This however, is not a realistic model as digital signatures do not provide any confidentiality, so one had to also ensure that the attacker knew all public keys, so that he was able to decrypt all signed values. A much nicer model is the one we have chosen in LySA^{XP}, where we distinguish between encryptions and signatures. An attacker can then learn anything that is signed even though he does not know the public key, but still he cannot learn anything encrypted if he does not know the key.

10.2.2 Processes

Processes are build from patterns and terms using the grammar in Table 10.3. We see that the syntax for processes in the LySA^{XP}-calculus are much alike the syntax for processes in the LySA-calculus.

Instead of the decryption operations in the LySA-calculus, we use the matching process as described in the Section 10.1:

$$\mathsf{match}\ t\ \mathsf{as}\ p.\ P'$$

P	::=	0	terminated process
	|	$P_1 \mid P_2$	parallel composition
	|	$!P$	replication
	|	$(\nu^T\ m)\, P$	restriction
	|	$\langle t \rangle.P$	output
	|	$(p).P$	input
	|	match t as p. P	pattern matching
T	::=	ε	general name
	|	$\pm$	public/private keypair

Table 10.3: Object-level processes; P

Which matches the term t against a pattern p and if this succeeds the remaining computation P' continues; eg. decryption in a symmetric key crypto-system, where the variable y should be decrypted with the secret key K and the decrypted content should be bound to the variable x, proceeds as follows:

$$\text{match } y \text{ as } \mathsf{E}_K(\_\%x)\,.$$

However, in general the pattern matching allows such decryptions to be done already in the input process, and hence the above decryption could be done as soon as the message was received.

$$(\mathsf{E}_K(\_\%x)\,).$$

where the value received is matched, ie. decrypted and the variable x is bound to the content. This example shows that the matching process should only be needed, if a value received cannot be matched until later in the protocol.

Also notice that restriction of names and keys for asymmetric cryptography is merged into one operation with a superscript T stating whether the restriction is a general name; n^{ε}, or an asymmetric key-pair; $n^{\pm}$.

10.3 Semantics

Having presented the formal syntax of LySA$^{\text{XP}}$ we shall now employ a reduction semantics. Following the design again, we shall begin with the semantics of basic elements in the calculus and then explain the semantics of the object-level.

10.3.1 Terms and Patterns

The semantics of terms will just be values ranged over $v \in \mathsf{Val}$, which are terms with no variables.

The semantics of pattern matching requires a bit more thought. Because of the binding construct, we must produce an environment θ for recording the variables bindings. We write [] for the empty environment and $\theta[x \mapsto v]$ for the environment that is as θ, except that it maps x to v. The semantics of pattern matching is given in Table 10.4 and takes the form:

$$\theta \vdash v \triangleright p : \theta'$$

which should be read as follows; in the environment θ, which contains all mappings of variables to their values in the pattern p, a matching of a value v against the pattern p produces the new environment θ'.

$$(\mathsf{Name}^{\mathsf{XP}})\ \theta \vdash n \triangleright n : \theta \qquad (\mathsf{SDec}^{\mathsf{XP}})\ \frac{\theta \vdash v_0 \triangleright p_0 : \theta' \quad \theta' \vdash v \triangleright p : \theta''}{\theta \vdash \mathsf{E}_{v_0}(v) \triangleright \mathsf{E}_{p_0}(p) : \theta''}$$

$$(\mathsf{Var}^{\mathsf{XP}})\ \theta \vdash v \triangleright x : \theta \text{ if } v = \theta x \qquad (\mathsf{PDec}^{\mathsf{XP}})\ \frac{\theta \vdash m^- \triangleright p_0 : \theta' \quad \theta' \vdash v \triangleright p : \theta''}{\theta \vdash \mathsf{P}_{m^+}(v) \triangleright \mathsf{P}_{p_0}(p) : \theta''}$$

$$(\mathsf{Wild}^{\mathsf{XP}})\ \theta \vdash v \triangleright \_ : \theta \qquad (\mathsf{UBli1}^{\mathsf{XP}})\ \frac{\theta \vdash v_0 \triangleright p_0 : \theta' \quad \theta' \vdash v \triangleright p : \theta''}{\theta \vdash \mathsf{B}_{v_0}(v) \triangleright \mathsf{B}_{p_0}(p) : \theta''}$$

$$(\mathsf{Bind}^{\mathsf{XP}})\ \frac{\theta \vdash v \triangleright p : \theta'}{\theta \vdash v \triangleright p\%x : \theta'[x \mapsto v]} \qquad (\mathsf{UBli2}^{\mathsf{XP}})\ \frac{\theta \vdash v_0 \triangleright p_0 : \theta' \quad \theta' \vdash \mathsf{S}_{m^-}(v) \triangleright p : \theta''}{\theta \vdash \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v)) \triangleright \mathsf{B}_{p_0}(p) : \theta''}$$

$$(\mathsf{Hash}^{\mathsf{XP}})\ \frac{\theta \vdash v \triangleright p : \theta'}{\theta \vdash \mathsf{H}(v) \triangleright \mathsf{H}(p) : \theta'} \qquad (\mathsf{Sign}^{\mathsf{XP}})\ \frac{\theta \vdash m^+ \triangleright p_0 : \theta' \quad \theta' \vdash v \triangleright p : \theta''}{\theta \vdash \mathsf{S}_{m^-}(v) \triangleright \mathsf{S}_{p_0}(p) : \theta''}$$

$$(\mathsf{Tup}^{\mathsf{XP}})\ \frac{\wedge_{i=1}^{k} \theta_{i-1} \vdash v_i \triangleright p_i : \theta_i}{\theta_0 \vdash \mathsf{T}(v_1, \cdots, v_k) \triangleright \mathsf{T}(p_1, \cdots, p_k) : \theta_k}$$

Table 10.4: Semantics of pattern matching; $\theta \vdash v \triangleright p : \theta'$.

The semantics of pattern matching of names ($\mathsf{Name}^{\mathsf{XP}}$) and wildcards ($\mathsf{Wild}^{\mathsf{XP}}$) are trivial, while pattern matching a value v against a variable x succeeds only if the value is in the environment of x; note that we write θx for the set of values

x maps to in θ. The most interesting rule is ($\mathsf{Bind}^{\mathsf{XP}}$) where the environment is updated; pattern matching a variable v against the pattern $p\%x$ in the environment θ succeeds if v matches p, hereby producing a new environment θ', where the variable x is bound to the value v. Cryptographic operations such as symmetric and asymmetric encryption, require that the key is known before matching the content of the encrypted value. .

10.3.2 Processes

The object-level semantics describes how a process P evolves to P' in a step by step fashion, formalised by a reduction relation $P \rightarrow P'$. This is defined as the smallest relation satisfying the rules in Table 10.5.

$$(\mathsf{Par}^{\mathsf{XP}}) \quad \frac{P_1 \rightarrow P_1'}{P_1 \mid P_2 \rightarrow P_1' \mid P_2} \qquad (\mathsf{New}^{\mathsf{XP}}) \quad \frac{P_1 \rightarrow P_1'}{(\nu^T\ m)\,P_1 \rightarrow (\nu^T\ m)\,P_1'}$$

$$(\mathsf{Com}^{\mathsf{XP}}) \quad \frac{\emptyset \vdash v \triangleright p : \theta}{\langle v \rangle.P_1 \mid (p).P_2 \rightarrow P_1 \mid P_2\theta}$$

$$(\mathsf{Con}^{\mathsf{XP}}) \quad \frac{P \equiv Q \quad Q \rightarrow Q' \quad Q' \equiv P'}{P \rightarrow P'} \qquad (\mathsf{Match}^{\mathsf{XP}}) \quad \frac{\emptyset \vdash v \triangleright p : \theta}{\mathsf{match}\ v\ \mathsf{as}\ p.\ P \rightarrow P\theta}$$

Table 10.5: Reduction relation for processes; $P \rightarrow P'$.

In the reduction relation we require the processes to be in a specific form to match the rules, and thus we shall use structural congruence ($\mathsf{Con}^{\mathsf{XP}}$) to syntactically manipulate the processes to be on this form. Structural congruence is defined as the smallest relation satisfying the rules in Table 10.6.

The rules for the structural congruence are standard, and as for LySA we also use α-equivalence. The rules for α-equivalence are given in Table 10.7, and two processes are α-equivalent if they only differ in the naming of bound names. The procedure of replacing bound names in a process with another name is again called α-conversion and is used in the object-level semantics. Note that α-converting a name preserves the tag τ of the name. Substitutions are extended homomorphically to terms, patterns and processes, for which we write $P\theta$ for a process P except that all free variables that are defined by θ are replaced by their values.

$$
\begin{array}{ll}
P \equiv P & P_1 \mid P_2 \equiv P_2 \mid P_1 \\
P_1 \equiv P_2 \Rightarrow P_2 \equiv P_1 & (P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3) \\
P_1 \equiv P_2 \wedge P_2 \equiv P_3 \Rightarrow P_1 \equiv P_3 & P \mid \mathsf{0} \equiv P \\
P_1 \equiv P_2 \Rightarrow P_1 \mid P_3 \equiv P_2 \mid P_3 & !P \equiv P \mid !P \\
P_1 \equiv P_2 \Rightarrow (\nu^T\ m)\, P_1 \equiv (\nu^T\ m)\, P_2 & P_1 \equiv P_2 \Rightarrow !P_1 \equiv !P_2 \\
(\nu^T\ m)\, \mathsf{0} \equiv \mathsf{0} & \\
(\nu^{T_1}\ m_1)\, (\nu^{T_2}\ m_2)\, P \equiv (\nu^{T_2}\ m_2)\, (\nu^{T_1}\ m_1)\, P & \\
(\nu^T\ m)\, (P_1 \mid P_2) \equiv P_1 \mid (\nu^T\ m)\, P_2 \quad \text{if } \{m^\tau \mid \tau \in T\} \cap \mathsf{fn}(P_1) = \emptyset & \\
P_1 \equiv P_2 \quad \text{if } P_1 \text{ and } P_2 \text{ are } \alpha\text{-equivalent} &
\end{array}
$$

Table 10.6: Structural congruence for processes; $P \equiv P'$.

$$
\begin{array}{ll}
P \overset{\alpha}{\equiv} P & \\
P_1 \overset{\alpha}{\equiv} P_2 \Rightarrow P_2 \overset{\alpha}{\equiv} P_1 & \\
P_1 \overset{\alpha}{\equiv} P_2 \wedge P_2 \overset{\alpha}{\equiv} P_3 \Rightarrow P_1 \overset{\alpha}{\equiv} P_3 & \\
 & \\
(\nu^\varepsilon\ n_1)\, P \overset{\alpha}{\equiv} (\nu^\varepsilon\ n_2)\, (P[n_1^\varepsilon \mapsto n_2^\varepsilon]) & \text{if } n_2^\varepsilon \notin \mathsf{fn}(P) \\
(\nu^\pm\ m_1)\, P \overset{\alpha}{\equiv} (\nu^\pm\ m_2)\, (P[m_1^+ \mapsto m_2^+, m_1^- \mapsto m_2^-]) & \text{if } m_1^+, m_1^-, m_2^+, m_2^- \notin \mathsf{fn}(P)
\end{array}
$$

Table 10.7: α-equivalence; $P \overset{\alpha}{\equiv} P'$.

Turning back to the reduction relation rules in Table 10.5, we have four remaining rules. The ($\mathsf{Com}^{\mathsf{XP}}$) rule states that if a process P_1' outputs a value v on the ether, a parallel process P_2' can input v and match it against a pattern p, update the environment with respect to the pattern matching and continue computing P_2. The rules for parallel processes ($\mathsf{Par}^{\mathsf{XP}}$) and restriction ($\mathsf{New}^{\mathsf{XP}}$) are straightforward whereas the rule for matching ($\mathsf{Match}^{\mathsf{XP}}$) is analogous to the communication rule.

Chapter 11

Modelling Protocols in LySA^{XP}

In Chapter 10 we presented the syntax and semantics for a new process calculus, for which one of the goals was that specifying protocols should be easier. In the following we will give the LySA^{XP} specification for the FOO92 protocol to illustrate that this goal is obtained.

11.1 The FOO92 protocol

Using the LySA^{XP}-calculus we shall now model the FOO92 voting protocol. As a reminder the protocol narration for FOO92, is relisted in Table 11.1 now using the LySA^{XP} notation.

In the first part of the thesis, the translation from the ordinary protocol narration into a LySA specification had an intermediate step, the extended protocol narration. As the LySA^{XP} specification does not need any calculus specific encoding tricks, we can immediately translate the extended narration to a specification of the FOO92 protocol. The LySA^{XP} specification of the FOO92 protocol is given in Table 11.2.

The first message exchange is quite similar to the original LySA specification,

1.	$V \rightarrow A$	:	$V,\ \mathsf{S}_{V^-}(\mathsf{B}_b(\mathsf{E}_r(v)))$	Preparation Phase
2.	$A \rightarrow V$	:	$\mathsf{S}_{K_A^-}(\mathsf{B}_b(\mathsf{E}_r(v)))$	Administration Phase
3.	$(V) \rightarrow C$	:	$\mathsf{S}_{K_A^-}(\mathsf{E}_r(v))$	Voting Phase
4.	$C \rightarrow$	:	$l,\ \mathsf{S}_{K_A^-}(\mathsf{E}_r(v))$	Publishing Phase
5.	$(V) \rightarrow C$	:	$l,\ r$	Opening Phase

Table 11.1: Protocol Narration for FOO92

```
      (ν± K_V) (ν± K_A)
      (   (ν^ε b) (ν^ε r) (ν^ε v)                          /* Voter */
1.        ⟨T(V, A, V, S_{K_V^-}(B_b(E_r(v))))⟩.
2'.       (T(A, V, S_{K_A^+}(B_b(E_r(v))) %x_1)).
2''.      match x_1 as B_b(_%x_2).
3.        ⟨T(D, C, x_2)⟩.
4'.       (T(C, D, _%x_3, x_2)).
5.        ⟨T(D, C, x_3, r)⟩.
      |
1'.       (T(V, A, V, S_{K_V^+}(_%y))).          /* Administrator */
2.        ⟨T(A, V, S_{K_A^-}(y))⟩.
      |
          (ν^ε l)                                        /* Counter */
3'.       (T(D, C, S_{K_A^+}(_%z_1) %z_2)).
4.        ⟨T(C, D, l, z_2)⟩.
5'.       (T(D, C, l, _%z_3)).
5''.      match z_1 as E_{z_3}(_%z_4).
      )
```

Table 11.2: FOO92 in LySA^{XP}-calculus

except that the signature check is performed already on input in step $1'$.

Message exchange 2 in the narration, was in the LySA specification done using a hack to specify the actual intention of the unblinding. In the LySA^{XP} specification, we shall divide this message exchange into three steps. The first step (step 2) is identical to step 2 in the original LySA specification; the administrator sends out the signed ballot. In the second step (step $2'$), we utilise that LySA^{XP} supports rebinding of variables, and hence we can check that the blinded ballot received is signed by the administrator, without removing the signature. In the last step (step $2''$), the variable x_1, that the signed ballot is bound to, is unblinded; thereby obtaining the valid ballot. Note that the voter does not

need to use the administrators secret key in this step, as he did in the LySA specification.

Message exchange 2 was divided into three steps to clarify that multiple checks and operations was made by the voter, but we can in fact express steps $2'$ and $2''$ in one step; 2^*, utilising the semantics of blinding:

$$2^*. \qquad (\mathsf{T}(A, V, \mathsf{B}_b(\mathsf{S}_{K_A^+}(\mathsf{E}_r(v))\,\%x_1))).$$

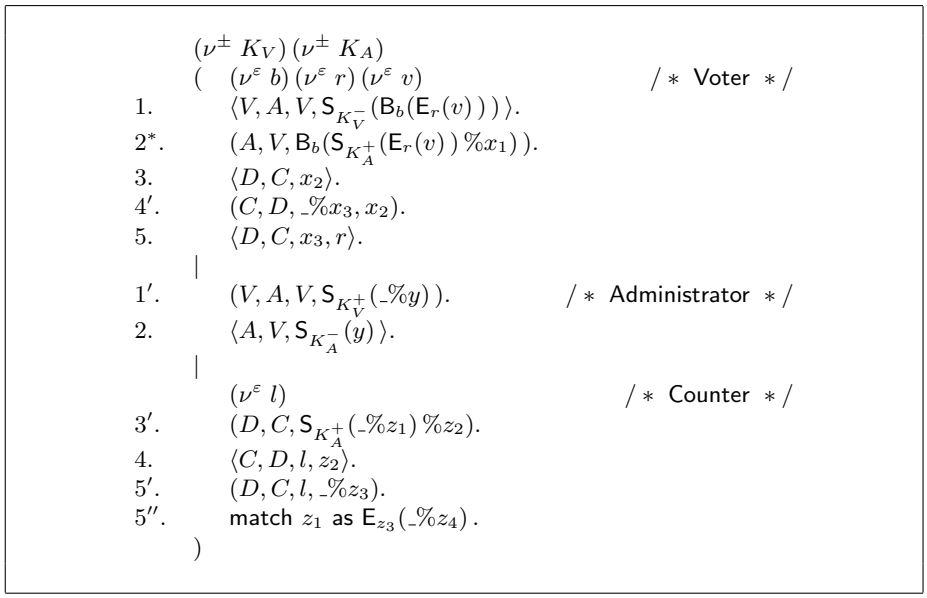
Observe that here the voter unblinds first, then pattern match on the signed ballot before finally assigning it to the variable x_1.

The third message exchange in the protocol is as in the LySA specification, but with signature check on the input to simplify the specification.

The specification of the fourth message exchange uses a strength of the LySA$^{\text{XP}}$ -calculus, that pattern matching is not restricted to the first j messages in a tuple, and hence the order from the original protocol narration can be used.

Specification of message exchange 5 is similar to the original LySA-calculus, except that we use the match construct instead of decrypting.

Previously we suggested that in an implementation of the calculus, the tuple construct could be internal of the analysis, and added in the parser. If we furthermore use the compact form of the second message, the LySA$^{\text{XP}}$ specification of the FOO92 protocol could be as simple as showed in Table 11.3.



$(\nu^{\pm}\ K_V)\,(\nu^{\pm}\ K_A)$
$($ $(\nu^{\varepsilon}\ b)\,(\nu^{\varepsilon}\ r)\,(\nu^{\varepsilon}\ v)$ /* Voter */
1. $\langle V, A, V, \mathsf{S}_{K_V^-}(\mathsf{B}_b(\mathsf{E}_r(v))\,)\rangle$.
2*. $(A, V, \mathsf{B}_b(\mathsf{S}_{K_A^+}(\mathsf{E}_r(v))\,\%x_1)\,)$.
3. $\langle D, C, x_2\rangle$.
4′. $(C, D, \_\%x_3, x_2)$.
5. $\langle D, C, x_3, r\rangle$.
|
1′. $(V, A, V, \mathsf{S}_{K_V^+}(\_\%y)\,)$. /* Administrator */
2. $\langle A, V, \mathsf{S}_{K_A^-}(y)\,\rangle$.
|
$(\nu^{\varepsilon}\ l)$ /* Counter */
3′. $(D, C, \mathsf{S}_{K_A^+}(\_\%z_1)\,\%z_2)$.
4. $\langle C, D, l, z_2\rangle$.
5′. $(D, C, l, \_\%z_3)$.
5″. match z_1 as $\mathsf{E}_{z_3}(\_\%z_4)$.
$)$

Table 11.3: FOO92 in LySA^{XP}-calculus with tuple-construct handled by parser

Chapter 12

Analysis of LySA^{XP}

We have seen that specifying protocols in the LySA^{XP}-calculus makes the translation, from the ordinary protocol narration into a mathematical model of the protocol, an easier task than it was with the original LySA-calculus. This leads us to the next step in our framework strategy; the analysis, which we present in the following sections.

12.1 Control Flow Analysis

We specify the analysis using flow logics as we did in the analysis of LySA. The aim of the analysis is to give a safe over-approximation of all possible value bindings of the variables, together with all the possible messages communicated on the ether. In the presentation of the LySA^{XP}-calculus in the previous chapters, we upheld the differentiation between the levels of description. Continuing this approach, we shall begin by presenting the analyses of the basic elements, and then use this as a basis for the analysis of processes.

In the analysis of the original LySA-calculus, we employed environments to keep track on potential variable bindings and all messages sent on the network. We shall use the same idea for LySA^{XP} and introduce a global abstract environment

ρ for potential variable bindings, and the global network environment κ for potential messages on the ether. These are similar to the environments used in the analysis of LySA, however instead of holding sequences of values, κ is just defined as a set of values, as we have the tuple construct for sequences.

12.1.1 Terms and Patterns

We shall handle the infinities of the sets as we did in LySA, by partitioning values into equivalence classes. Each class is assigned a representative; a canonical value, and hence the analysis of a term t without variables result in a singular canonical value. However, when a term includes variables, it may describe a larger set of canonical values. Therefore we need a way to evaluate the set $\rho[\![t]\!] \subseteq \lfloor \mathsf{Val} \rfloor$ of canonical values that a term may represent in environment ρ. The definition of this evaluation of a term is given in Table 12.1.

$$
\begin{array}{lcl}
\rho[\![n]\!] & \stackrel{\text{def}}{=} & \{\lfloor n \rfloor\} \\
\rho[\![x]\!] & \stackrel{\text{def}}{=} & \rho(\lfloor x \rfloor) \\
\rho[\![\mathsf{T}(t_1, \cdots, t_k)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{T}(v_1, \cdots, v_k) \,|\, v_1 \in \rho[\![t_1]\!], \cdots, v_k \in \rho[\![t_k]\!]\} \\
\rho[\![\mathsf{H}(t)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{H}(v) \,|\, v \in \rho[\![t]\!]\} \\
\rho[\![\mathsf{E}_{t_0}(t)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{E}_{v_0}(v) \,|\, v_0 \in \rho[\![t_0]\!], v \in \rho[\![t]\!]\} \\
\rho[\![\mathsf{P}_{t_0}(t)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{P}_{v_0}(v) \,|\, v_0 \in \rho[\![t_0]\!], v \in \rho[\![t]\!]\} \\
\rho[\![\mathsf{B}_{t_0}(t)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{B}_{v_0}(v) \,|\, v_0 \in \rho[\![t_0]\!], v \in \rho[\![t]\!]\} \\
\rho[\![\mathsf{S}_{t_0}(t)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{S}_{v_0}(v) \,|\, v_0 \in \rho[\![t_0]\!], v \in \rho[\![t]\!]\}
\end{array}
$$

Table 12.1: Evaluating a term; $\rho[\![t]\!]$.

The definition is straightforward as names are evaluated to their canonical representative and variables are evaluated to the set of canonical values they are bound to in ρ. Evaluating a compound construct proceeds by recursively evaluating each of the terms in the construct.

Analysis of pattern matching can be divided into two issues; the first issue concerns the possible variable bindings in the matching, and the second issue concerns the actual pattern matching with respect to the variable bindings. The general idea of the analysis is then, that each time a pattern matching is to be made, the analysis should first ensure that the environment ρ holds all possible

variable bindings, before performing the actual pattern matching. Hence the analysis of pattern matching is divided into two forms of judgements:

$$\rho \models v \triangleright p$$

which returns true if the pattern matching of a value v against a pattern p succeeds and false otherwise, and

$$\rho \models v \sim p$$

which returns true if and only if all possible variable bindings that may occur during the matching of v against p, are in ρ. Note that the value v must be a canonical value.

The analysis of pattern matching is given in Table 12.2 while the analysis of bindings in patterns is presented in Table 12.3, and these are explained in detail below.

$(\mathsf{AN}^{\mathsf{XP}})$	$\rho \models v \triangleright n$	iff	$v = \lfloor n \rfloor$
$(\mathsf{AVar}^{\mathsf{XP}})$	$\rho \models v \triangleright x$	iff	$v \in \rho(\lfloor x \rfloor)$
$(\mathsf{AWild}^{\mathsf{XP}})$	$\rho \models v \triangleright \_$	iff	true
$(\mathsf{ABind}^{\mathsf{XP}})$	$\rho \models v \triangleright p\%x$	iff	$\rho \models v \triangleright p$
$(\mathsf{ATup}^{\mathsf{XP}})$	$\rho \models v \triangleright \mathsf{T}(p_1, \cdots, p_k)$	iff	$v = \mathsf{T}(v_1, \cdots, v_k) \ \wedge \wedge_{i=1}^{k} \rho \models v_i \triangleright p_i$
$(\mathsf{AHash}^{\mathsf{XP}})$	$\rho \models v \triangleright \mathsf{H}(p)$	iff	$v = \mathsf{H}(v') \ \wedge \ \rho \models v' \triangleright p$
$(\mathsf{AEnc}^{\mathsf{XP}})$	$\rho \models v \triangleright \mathsf{E}_{p_0}(p)$	iff	$v = \mathsf{E}_{v_0}(v') \ \wedge \ \rho \models v_0 \triangleright p_0 \ \wedge \ \rho \models v' \triangleright p$
$(\mathsf{APub}^{\mathsf{XP}})$	$\rho \models v \triangleright \mathsf{P}_{p_0}(p)$	iff	$v = \mathsf{P}_{m^+}(v') \ \wedge \ \rho \models m^- \triangleright p_0 \ \wedge \ \rho \models v' \triangleright p$
$(\mathsf{ABli}^{\mathsf{XP}})$	$\rho \models v \triangleright \mathsf{B}_{p_0}(p)$	iff	$(v = \mathsf{B}_{v_0}(v') \ \wedge \ \rho \models v_0 \triangleright p_0 \ \wedge \ \rho \models v' \triangleright p)$ $\vee$ $(v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))\ \wedge$ $\rho \models v_0 \triangleright p_0 \ \wedge \ \rho \models \mathsf{S}_{m^-}(v') \triangleright p)$
$(\mathsf{ASign}^{\mathsf{XP}})$	$\rho \models v \triangleright \mathsf{S}_{p_0}(p)$	iff	$v = \mathsf{S}_{m^-}(v') \ \wedge \ \rho \models m^+ \triangleright p_0 \ \wedge \ \rho \models v' \triangleright p$

Table 12.2: Analysis of pattern matching; $\rho \models v \triangleright p$.

The analysis of pattern matching is straightforward. As binding of variables is handled by the analysis $\rho \models v \sim p$, this analysis merely concerns the matching of a value v against the pattern p in the environment ρ. The case for matching against a name n $(\mathsf{AN}^{\mathsf{XP}})$ checks if the value v is the canonical name of n. Similarly in the case for match against a variable $(\mathsf{AVar}^{\mathsf{XP}})$ where the analysis returns true, if v is in the set of canonical values that x evaluates to; $\rho(\lfloor x \rfloor)$. The rest of the cases are defined recursively in the structure of terms and requires that all sub-terms must pattern match. Notice that in the case for blinding

$$
\begin{array}{lll}
(\mathsf{ABN}^{\mathsf{XP}}) & \rho \models v \sim n & \text{iff } \mathsf{true} \\
(\mathsf{ABVar}^{\mathsf{XP}}) & \rho \models v \sim x & \text{iff } \mathsf{true} \\
(\mathsf{ABWild}^{\mathsf{XP}}) & \rho \models v \sim \_ & \text{iff } \mathsf{true} \\
(\mathsf{ABBind}^{\mathsf{XP}}) & \rho \models v \sim p\%x & \text{iff } \rho \models v \sim p \wedge (\rho \models v \triangleright p \Rightarrow v \in \rho(\lfloor x \rfloor)) \\
(\mathsf{ABTup}^{\mathsf{XP}}) & \rho \models v \sim \mathsf{T}(p_1, \cdots, p_k) & \text{iff } v = \mathsf{T}(v_1, \cdots, v_k) \Rightarrow \\
 & & \quad \rho \models v_1 \sim p_1 \wedge (\rho \models v_1 \triangleright p_1 \Rightarrow \\
 & & \quad\quad \rho \models v_2 \sim p_2 \wedge (\rho \models v_2 \triangleright p_2 \Rightarrow \\
 & & \quad\quad\quad \cdots \rho \models v_k \sim p_k) \cdots) \\
(\mathsf{ABHash}^{\mathsf{XP}}) & \rho \models v \sim \mathsf{H}(p) & \text{iff } \mathsf{true} \\
(\mathsf{ABEnc}^{\mathsf{XP}}) & \rho \models v \sim \mathsf{E}_{p_0}(p) & \text{iff } v = \mathsf{E}_{v_0}(v') \Rightarrow \rho \models v_0 \triangleright p_0 \Rightarrow \rho \models v' \sim p \\
(\mathsf{ABPub}^{\mathsf{XP}}) & \rho \models v \sim \mathsf{P}_{p_0}(p) & \text{iff } v = \mathsf{P}_{m^+}(v') \Rightarrow \rho \models m^+ \triangleright p_0 \Rightarrow \rho \models v' \sim p \\
(\mathsf{ABBli}^{\mathsf{XP}}) & \rho \models v \sim \mathsf{B}_{p_0}(p) & \text{iff } (v = \mathsf{B}_{v_0}(v') \Rightarrow \rho \models v_0 \triangleright p_0 \Rightarrow \rho \models v' \sim p) \\
 & & \quad \vee \\
 & & \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))\ \Rightarrow \\
 & & \quad\quad \rho \models v_0 \triangleright p_0 \Rightarrow \rho \models \mathsf{S}_{m^-}(v') \sim p) \\
(\mathsf{ABSign}^{\mathsf{XP}}) & \rho \models v \sim \mathsf{S}_{p_0}(p) & \text{iff } v = \mathsf{S}_{m^-}(v') \Rightarrow \rho \models v_0 \triangleright p_0 \Rightarrow \rho \models v' \sim p
\end{array}
$$

Table 12.3: Bindings in pattern matching; $\rho \models v \sim p$.

($\mathsf{ABLi}^{\mathsf{XP}}$), the two rules for unblinding has been merged together and analysis requires that the value v is either a blinded value or a signed blinded value.

The analysis of bindings in a pattern matching only concerns the possible bindings of variables in the matching, and hence the cases for names, variables and wildcards should just return true. The case where the bindings actually takes place is ($\mathsf{ABBind}^{\mathsf{XP}}$), which proceeds as follows; first all possible variable bindings in the pattern matching of v against the pattern p should be made, then if the pattern matching of v against p is successful in the updated environment ρ, then ρ is also updated with the binding of the variable x to v. Notice that also here the procedure of first ensuring that all variables are bound before performing the pattern matching is used.

Another interesting case is the tuple construct ($\mathsf{ABTup}^{\mathsf{XP}}$); here we make all the possible bindings in the pattern matching from left to right. This means that matching a tuple $\mathsf{T}(v_1, v_2, \cdots, v_k)$ against a pattern $\mathsf{T}(p_1, p_2, \cdots, p_k)$ will first make all the possible bindings in pattern matching of v_1 against p_1 before performing the pattern matching, and only if the pattern matching succeeds we proceed with the next element in the tuple. The reason for this, is that if the pattern does not match in the j-th element of the tuple, we need not make any bindings in the remaining $k - j$ pattern matchings. Also notice that this ensures left-to-right association, which can be illustrated in the following example; consider a pattern matching of a tuple of a key K and a symmetric

encryption $\mathsf{E}_K(m)$. If the values in the tuple are ordered as $\mathsf{T}(K, \mathsf{E}_K(m))$, it can be matched against the pattern $\mathsf{T}(\_\%x, \mathsf{E}_x(\_\%y))$ because binding of K to x updates the environment with $[x \mapsto K]$ which is required for decryption, ie. matching $\mathsf{E}_K(m)$ against $\mathsf{E}_x(\_\%y)$. If the order of the tuple and the corresponding pattern is reversed; matching $\mathsf{T}(\mathsf{E}_K(m), K)$ against $\mathsf{T}(\mathsf{E}_x(\_\%y), \_\%x)$, then the matching is not successful.

The rest of the cases use the same approach, notice that we do not need to analyse for variable bindings in keys and hashed terms, as the syntax of constructive patterns and signature patterns, does not allow variable bindings.

12.1.2 Processes

The analysis of processes takes the form:

$$\rho, \kappa \models P$$

This means that the abstract environments ρ and κ are valid estimates for the process P. The analysis of processes is defined in Table 12.4 and is fairly straightforward.

($\mathsf{APar}^{\mathsf{XP}}$)	$\rho, \kappa \models P_1 \mid P_2$	iff	$\rho, \kappa \models P_1 \wedge \rho, \kappa \models P_2$
($\mathsf{ABang}^{\mathsf{XP}}$)	$\rho, \kappa \models !P$	iff	$\rho, \kappa \models P$
($\mathsf{ANil}^{\mathsf{XP}}$)	$\rho, \kappa \models 0$	iff	true
($\mathsf{ANew}^{\mathsf{XP}}$)	$\rho, \kappa \models (\nu^T\ m)\ P$	iff	$\rho, \kappa \models P$
($\mathsf{AOut}^{\mathsf{XP}}$)	$\rho, \kappa \models \langle t \rangle.P$	iff	$\rho[\![t]\!] \subseteq \kappa \wedge \rho, \kappa \models P$
($\mathsf{AIn}^{\mathsf{XP}}$)	$\rho, \kappa \models (p).P$	iff	$\forall v \in \kappa : \rho \models v \sim p \wedge ($ $\rho \models v \triangleright p \Rightarrow \rho, \kappa \models P)$
($\mathsf{AMat}^{\mathsf{XP}}$)	$\rho, \kappa \models \mathsf{match}\ t\ \mathsf{as}\ p.\ P$	iff	$\forall v \in \rho[\![t]\!] : \rho \models v \sim p \wedge ($ $\rho \models v \triangleright p \Rightarrow \rho, \kappa \models P)$

Table 12.4: Analysis of processes; $\rho, \kappa \models P$.

The analysis of the ($\mathsf{ANil}^{\mathsf{XP}}$) process always succeeds, and restrictions of names are ignored by the analysis ($\mathsf{ANew}^{\mathsf{XP}}$), as it was in the analysis of LySA. Similarly in the cases for replication ($\mathsf{ABang}^{\mathsf{XP}}$) and parallel composition ($\mathsf{APar}^{\mathsf{XP}}$) which are also analogous to the analysis of LySA.

Turning to the more interesting cases, the analysis of output ($\mathsf{AOut}^{\mathsf{XP}}$) uses the evaluation of terms, to ensure that all values the term t may evaluate to

are included in the analysis component κ, before continuing analysis of the subsequent process P.

The case for input ($\mathsf{AIn}^{\mathsf{XP}}$) makes use of the analysis of variable bindings in pattern matchings. For each value v on the ether; ie. in κ, the analysis ensures that all variable bindings that may arise from the pattern matching of v against p are added to the component ρ. Then the analysis pattern matches v against p, and only if this pattern matching succeeds, the analysis proceeds with the subsequent process P.

The rule for matching ($\mathsf{AMat}^{\mathsf{XP}}$) is identical to the one for input, but instead of matching all values on the ether, the analysis matches all values the term t may evaluate to.

The following example shows how the analysis works. To ease readability we have left out many of the computations, however the unabridged example can be found in Appendix B.1.

Example 12.1 Analysis of a LySA^{XP} protocol *Consider the following protocol in the LySA^{XP}-calculus*

$$
\begin{array}{lll}
P ::= & (\nu^{\varepsilon}\ B)\,(\nu^{\varepsilon}\ msg) & /*\,Client\,*/ \\
& \langle \mathsf{B}_B(msg)\,\rangle. & \\
& (\mathsf{B}_B(\_\%x)\,). & \\
& \langle x\rangle.\mathsf{0} & \\
\mid & (\nu^{\pm}\ K_S) & /*\,Server\,*/ \\
& (\_\%y). & \\
& \langle \mathsf{S}_{K_S^-}(y)\,\rangle.\mathsf{0} &
\end{array}
$$

The protocol proceeds as follows. First the client sends a blinded message to the server, then the server receives a message from the ether and returns the message with its signature. Next the client receives a message and if it is blinded with his own blinding factor B, it is unblinded and the result is bound to the variable x which is then sent back out on the ether.

The first step is to recursively apply the analysis of processes:

$$
\begin{array}{lll}
\rho,\kappa \models P & \mathsf{iff} & (\rho[\![\mathsf{B}_B(msg)]\!] \subseteq \kappa \wedge (\\
& & \quad \forall v \in \kappa : \rho \models v \sim \mathsf{B}_B(\_\%x) \ \wedge (\\
& & \quad\quad \rho \models v \triangleright \mathsf{B}_B(\_\%x) \Rightarrow (\\
& & \quad\quad \rho[\![x]\!] \subseteq \kappa \wedge \mathsf{true})))) \\
& & \wedge \\
& & (\forall v \in \kappa : \rho \models v \sim \_\%y \wedge (\\
& & \quad \rho \models v \triangleright \_\%y \Rightarrow (\\
& & \quad\quad \rho[\![\mathsf{S}_{K_S^-}(y)]\!] \subseteq \kappa \wedge \mathsf{true})))
\end{array}
$$

We then apply the analysis of evaluation of terms, analysis of bindings and pattern matching:

$$
\begin{array}{lll}
\rho,\kappa \models P & \text{iff} & (\{\mathsf{B}_{\lfloor B\rfloor}(\lfloor msg\rfloor)\} \subseteq \kappa \wedge \\
& & \quad (\forall v \in \kappa : \\
& & \quad ((v = \mathsf{B}_{v_0}(v') \Rightarrow (v_0 = \lfloor B\rfloor \Rightarrow \\
& & \qquad (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v' \in \rho(x))))) \vee \\
& & \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \Rightarrow \\
& & \qquad (v_0 = \lfloor B\rfloor \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x))))) \wedge \\
& & \qquad (((v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B\rfloor \wedge \mathsf{true}) \vee \\
& & \qquad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \wedge v_0 = \lfloor B\rfloor \wedge \mathsf{true})) \Rightarrow \\
& & \qquad\quad (\rho(\lfloor x\rfloor) \subseteq \kappa \wedge \mathsf{true})))) \\
& & \wedge \\
& & (\forall v \in \kappa : (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v \in \rho(y))) \wedge \\
& & \quad (\mathsf{true} \Rightarrow (\{\mathsf{S}_{\lfloor K_S^-\rfloor}(v) \,|\, v \in \rho(\lfloor y\rfloor)\} \subseteq \kappa \wedge \mathsf{true})))
\end{array}
$$

By using simple logics we can reduce this rather complex constraint, thereby obtaining the following constraint.

$$
\begin{array}{lll}
\rho,\kappa \models P & \text{iff} & \{\mathsf{B}_{\lfloor B\rfloor}(\lfloor msg\rfloor)\} \subseteq \kappa \wedge \\
& & (\forall v \in \kappa : (v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B\rfloor) \Rightarrow \\
& & \quad (v' \in \rho(x) \wedge \rho(\lfloor x\rfloor) \subseteq \kappa)) \wedge \\
& & (\forall v \in \kappa : (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \wedge v_0 = \lfloor B\rfloor) \Rightarrow \\
& & \quad (\mathsf{S}_{m^-}(v') \in \kappa\kappa\rho(x) \wedge \rho(\lfloor x\rfloor) \subseteq \kappa)) \wedge \\
& & (\kappa \subseteq \rho(y)) \wedge \\
& & (\{\mathsf{S}_{\lfloor K_S^-\rfloor}(v) \,|\, v \in \rho(\lfloor y\rfloor)\} \subseteq \kappa)
\end{array}
$$

Which can relatively easy be seen to have the following smallest solution:

$$
\begin{array}{lcl}
\kappa & = & \{\mathsf{B}_{\lfloor B\rfloor}(\lfloor msg\rfloor)\} \cup \{\mathsf{S}_{\lfloor K_S^-\rfloor}(v) \,|\, v \in \kappa\} \cup \rho(x) \\
\rho(x) & = & \{\mathsf{S}_{\lfloor K_S^-\rfloor}(\lfloor msg\rfloor), \lfloor msg\rfloor\} \\
\rho(y) & = & \kappa
\end{array}
$$

Notice that κ holds a recursive element and is therefore an infinite set. □

12.2 Soundness of the Analysis

In this section we shall prove the soundness of the analysis of LySA$^{\text{XP}}$; ie. the analysis of processes from above correctly captures the formal semantics of Table 10.5.

Even though LySA^{XP} differs from LySA in several ways, we can apply the same proof technique here as the one we used for the analysis of LySA in Chapter 4. This means that we shall prove a subject reduction lemma, which states that the analysis $\rho, \kappa \models P$ captures any behavior of the process P, and use this result to show that the analysis components ρ and κ safely approximate all variable bindings and all messages sent on the network respectively. We shall also present a number of lemmata similar to those in the soundness proof for LySA, which illustrate that the analysis of LySA^{XP} has the same properties as the analysis of LySA.

The first lemma shows that the analysis can only distinguish names that belong to different equivalence classes; ie. has different canonical representatives.

Lemma 12.1 (Invariance of canonical names) *If* $\rho, \kappa \models P$ *and* $\lfloor n \rfloor = \lfloor n' \rfloor$ *then* $\rho, \kappa \models P[n \mapsto n']$.

Proof The lemma is a direct consequence of the fact that the analysis only records canonical names. The proof proceeds straightforward by induction in the definition of the analysis, with the only interesting case being the rule ($\mathsf{AN}^{\mathsf{XP}}$) though it too is straightforward as $\lfloor n \rfloor = \lfloor n[n \mapsto n'] \rfloor = \lfloor n \rfloor$ □

Now following the proof technique used for LySA, the next lemma shows that the analysis cannot tell α-equivalent processes apart. As for LySA we shall only consider the semantics using disciplined α-equivalence, defined as follows:

Definition 12.2 (Disciplined α-equivalence) Two processes P_1 and P_2 are *disciplined* α-equivalent whenever $P_1 \stackrel{\alpha}{\equiv} P_2$ using the rules in Table 10.7 with the extra requirement that $\lfloor n_1^{\varepsilon} \rfloor = \lfloor n_2^{\varepsilon} \rfloor$, $\lfloor m_1^{-} \rfloor = \lfloor m_2^{-} \rfloor$ and $\lfloor m_1^{+} \rfloor = \lfloor m_2^{+} \rfloor$.

Which leads us to the invariance of α-equivalence result.

Lemma 12.3 (Invariance of α-equivalence) *If* $\rho, \kappa \models P$ *and* P *is disciplined* α*-equivalent with* P' *then* $\rho, \kappa \models P'$.

Proof The proof proceeds by induction in the definition of α-equivalence in Table 10.7. The cases for the equivalence follow by the induction hypothesis. The remaining cases follow from Lemma 12.1 remembering that substituted names have the same canonical name as the substitute. □

And at last we have the general result of invariance of structural congruence.

Lemma 12.4 (Invariance of structural congruence) *If* $\rho, \kappa \models P$ *and* $P \equiv P'$ *then* $\rho, \kappa \models P'$.

Proof The proof proceeds by induction in the definition of $P \equiv P'$ defined in Table 10.6

Cases for equivalence and congruence follow by the induction hypothesis.

Cases for parallel composition follow because logical conjunction used in the analysis is commutative and associative. Furthermore, logical conjunction has true as a neutral element and true is equivalent to the analysis of $\mathsf{0}$, which is the neutral element of parallel composition.

Cases for replication Assume $\rho, \kappa \models !P$. Then the following calculation justifies that also $\rho, \kappa \models P \mid !P$:

$$\begin{array}{lllll} \rho, \kappa \models !P & \text{iff} & \rho, \kappa \models P & & (\mathsf{ABang}^{\mathsf{XP}}) \\ & \text{iff} & \rho, \kappa \models P \wedge \rho, \kappa \models P & & \\ & \text{iff} & \rho, \kappa \models P \wedge \rho, \kappa \models !P & & (\mathsf{ABang}^{\mathsf{XP}}) \\ & \text{iff} & \rho, \kappa \models P \mid !P & & (\mathsf{APar}^{\mathsf{XP}}) \end{array}$$

That $P_1 \equiv P_2$ implies $!P_1 \equiv !P_2$ follows directly from $(\mathsf{ABang}^{\mathsf{XP}})$.

Cases for restriction are straightforward to check using the fact that the analysis ignores restriction.

Case for α-equivalence follows from Lemma 12.3. □

This shows that the analysis of LySA$^{\mathsf{XP}}$ uses the same technique as LySA to restrict the analysis, such that it is efficiently computable.

Proceeding to the more interesting part, we are now posed with a problem as the analysis of pattern matching is separated into two analyses in LySA$^{\mathsf{XP}}$. The result we need is that if a value v may match a pattern p in the semantics, then if the analysis of variable bindings in the pattern matching holds then it implies that also the analysis of the matching holds. Or formally, that if $\emptyset \vdash v \triangleright p : \theta$ then it must be the case that $\rho \models v \sim p$ implies $\rho \models v \triangleright p$.

To prove this, we shall need two sub-results. First we need to prove that if v may match p in the semantics and ρ holds all possible variable bindings created in the match, then it must be the case that v matches p in the analysis. This is captured by the following lemma.

Lemma 12.5 (Matching of values) *If* $v \in \lfloor \mathsf{Val} \rfloor$, $\emptyset \vdash v \triangleright p : \theta$ *and* $\forall x : v' = \theta x \Rightarrow v' \in \rho(\lfloor x \rfloor)$ *then* $\rho \models v \triangleright p$.

Proof The proof is done by structural induction in the semantics of pattern matching as defined in Table 10.4, where each case follows directly from the induction hypothesis. □

Next we need to show that if the matching v to p will lead to the variable binding $v' = \theta x$, then the analysis of variable bindings in patterns $\rho \models v \sim p$ will lead to exactly this variable binding in ρ. In other words, that the analysis of variable bindings in pattern matchings captures all possible variable bindings in the semantics.

Lemma 12.6 (Binding of variables) *If* $v \in \lfloor \mathsf{Val} \rfloor$, $\emptyset \vdash v \triangleright p : \theta$ *and* $\rho \models v \sim p$ *it follows that if* $v' = \theta x$ *then* $v' \in \rho(\lfloor x \rfloor)$.

Proof The proof proceeds by structural induction in the semantics of pattern matching as defined in Table 10.4. In this case the interesting case is (Bind^{XP}).

Case (Name^{XP}), (Var^{XP}), (Wild^{XP}) follows directly from the induction hypothesis.

Case (Tup^{XP}), (Hash^{XP}), (SDec^{XP}), (PDec^{XP}), ($\mathsf{UBli1}^{XP}$), ($\mathsf{UBli2}^{XP}$), (Sign^{XP}) follows directly from the induction hypothesis and the fact that the syntax does not allow any bindings within hashings, in keys, in blinding factors or in signatures.

Case (Bind^{XP}) Let $v \in \lfloor \mathsf{Val} \rfloor$, $\emptyset \vdash v \triangleright p\%x : \theta$ and $\rho \models v \sim p\%x$, hence we have from (Bind^{XP}) that $\theta x = v$. Then by the induction hypothesis $\rho \models v \sim p$ which from the analysis allows us to conclude $v \in \rho(\lfloor x \rfloor)$. □

Now using these two lemmata, we are ready to present the result for pattern matching.

Lemma 12.7 (Evaluation of pattern matchings) *If* $v \in \lfloor \mathsf{Val} \rfloor$, $\emptyset \vdash v \triangleright p : \theta$ *and* $\rho \models v \sim p$ *then* $\rho \models v \triangleright p$.

Proof The proof follows directly from Lemma 12.5 and Lemma 12.6. □

Proceeding as for the soundness proof for LySA, we must show that the analysis components are resistant to substitutions

Lemma 12.8 (Substitution in processes) *If* $\rho, \kappa \models P$ *and* $\lfloor v \rfloor \in \rho(\lfloor x \rfloor)$ *then* $\rho, \kappa \models P[x \overset{\alpha}{\mapsto} v]$.

Proof The lemma follows from straightforward induction applying the induction hypothesis on any subprocesses. It relies on Lemma 12.3 because the analysis is invariant under any α-conversion that may occur due to capture avoiding substitution. □

In the semantics a process is not updated by a substitution, but by an entire environment. But this is analogue to multiple substitutions, hence we get the following simple lemma.

Lemma 12.9 (General substitution in processes) *If* $\rho, \kappa \models P$, $v \in \lfloor \mathsf{Val} \rfloor$, $\rho \models v \sim p$ *and* $\emptyset \vdash v \triangleright p : \theta$ *then* $\rho, \kappa \models P\theta$.

Proof Since $\theta = [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]$ the Lemma is just the result of repeatedly applying Lemma 12.8 to Lemma 12.6. □

We now only need a simple lemma before we can prove the subject reduction lemma.

Lemma 12.10 (Evaluation of values) *If* $v \in \lfloor \mathsf{Val} \rfloor$ *then* $[\![v]\!] = \{v\}$.

Proof The lemma follows directly from the definition of evaluation of terms in Table 12.1. □

We are now ready to present the subject reduction lemma, which shows that our analysis correctly captures any behavior of the protocol.

Lemma 12.11 (Subject Reduction) *If* $\rho, \kappa \models P$ *and* $P \to P'$ *then* $\rho, \kappa \models P'$.

Proof The proof proceeds by structural induction in the reduction steps.

Case ($\mathsf{Par}^{\mathsf{XP}}$) Assume that $\rho, \kappa \models P_1 \mid P_2$ i.e. that $\rho, \kappa \models P_1$ and $\rho, \kappa \models P_2$. Furthermore assume that $P_1 \mid P_2 \to P_1' \mid P_2$ by ($\mathsf{Par}^{\mathsf{XP}}$) because $P_1 \to P_1'$. Then using the induction hypothesis also $\rho, \kappa \models P_1'$. The analysis then allows to conclude that $\rho, \kappa \models P_1' \mid P_2$.

Case ($\mathsf{New}^{\mathsf{XP}}$) Assume that $\rho, \kappa \models (\nu^T\ m)\, P$ i.e. that $\rho, \kappa \models P$. Assume also that $(\nu^T\ m)\, P \to (\nu^T\ m)\, P'$ using ($\mathsf{New}^{\mathsf{XP}}$) because $P \to P'$. Then by the induction hypothesis $\rho, \kappa \models P'$ which by the analysis definition allows us to conclude $\rho, \kappa \models (\nu^T\ m)\, P'$.

Case (Com^{XP}) Let $v \in \lfloor \mathsf{Val} \rfloor$, $P = \langle v \rangle.P_1 \mid (p).P_2$ and $P' = P_1 \mid P_2\theta$ and assume that $\rho, \kappa \models P$. Also assume that $P \rightarrow P'$ and and that $\emptyset \vdash v \triangleright p : \theta$ due to (Com^{XP}). Expanding the analysis one gets

$$\begin{array}{lll} \rho, \kappa \models P & \text{iff} & \rho, \kappa \models \langle v \rangle.P_1 \mid (p).P_2 \\ & \text{iff} & \rho, \kappa \models \langle v \rangle.P_1 \wedge \rho, \kappa \models (p).P_2 \\ & \text{iff} & \rho[\![v]\!] \subseteq \kappa \wedge \\ & & \rho, \kappa \models P_1 \wedge \\ & & \forall v' \in \kappa : \rho \models v' \sim p \wedge (\rho \models v' \triangleright p \Rightarrow \rho, \kappa \models P_2) \end{array}$$

From the analysis of output and Lemma 12.10 one may conclude that $\rho, \kappa \models P_1$ and $v \in \kappa$. Using the latter, the analysis of the input gives that $\rho \models v \sim p$ and that $\rho \models v \triangleright p \Rightarrow \rho, \kappa \models P_2$. By applying Lemma 12.7 and since $\emptyset \vdash v \triangleright p : \theta$ one may conclude that $\rho \models v \triangleright p$ and hence that $\rho, \kappa \models P_2$. Finally from Lemma 12.9 one may conclude that $\rho, \kappa \models P_2\theta$ which also means that $\rho, \kappa \models P_1 \mid P_2\theta$ and thus $\rho, \kappa \models P'$.

Case (Con^{XP}) is a direct consequence of the induction hypothesis and application of Lemma 12.4

Case (Match^{XP}) is similar to (Com^{XP}). □

And again as for the soundness proof of the LySA analysis, the subject reduction result can be used to show that the analysis components are safe estimates. This is singled out by the following two theorems.

Theorem 12.12 (Messages in κ) *If $\rho, \kappa \models P$ and $P \rightarrow^* P' \rightarrow P''$ such that the reduction $P' \rightarrow P''$ is derived using* (Com^{XP}) *on output $\langle v \rangle.P_1''$ then $\lfloor v \rfloor \in \kappa$.*

Proof By induction in the length of the reduction sequence, Lemma 12.11 can be used to conclude that $\rho, \kappa \models P'$. Next the proof proceeds by induction in the reduction rules used to derive $P' \rightarrow P''$.

Case (Com^{XP}) If this rule is applied, it will be a process of the form

$$\langle v \rangle.P_1'' \mid (v).P_2''$$

The analysis holds for this process meaning, in particular, that the analysis of output holds for the communication of the value. Using Lemma 12.10 one can check that then indeed $\lfloor v \rfloor \in \kappa$.

Case ($\mathsf{Match}^{\mathsf{XP}}$). Reductions that uses any of these rules will not also use the rule ($\mathsf{Com}^{\mathsf{XP}}$) and can therefore be disregarded.

Case ($\mathsf{New}^{\mathsf{XP}}$), ($\mathsf{Par}^{\mathsf{XP}}$), ($\mathsf{Con}^{\mathsf{XP}}$) are all straightforward by appliying the induction hypothesis noting that the analysis also holds for any subprocesses. □

and

Theorem 12.13 (Values in ρ) *If $\rho, \kappa \models P$ and $P \rightarrow^* P' \rightarrow P''$ such that P'' is the result of a substitution of the variable x for the value v the $\lfloor v \rfloor \in \rho(\lfloor x \rfloor)$.*

Proof The proof is similar to that of Theorem 12.12. □

This shows that the analysis of LySA$^{\mathsf{XP}}$ is sound with respect to the operational semantics, and that the analysis components ρ and κ hold all possible variable bindings and all messages that may be sent on the ether, which is exactly the result we wanted.

12.3 The Attacker

We will now briefly discuss how to model the attacker for the analysis. Following the trend from LySA, we want to specify an attacker process to be used by the analysis in the setting

$$P \mid \bullet$$

where P is the protocol being analysed and $\bullet$ is the attacker. The attacker for LySA was in Chapter 12 defined using the Dolev-Yao approach, and we shall use the same approach in LySA$^{\mathsf{XP}}$.

The analysis of the attacker process is given in Table 12.5, here we have used the set $\mathcal{N}$ for all variables occurring free in P. The analysis shows that the attacker initially has some knowledge ($\mathsf{DY1}^{\mathsf{XP}}$), that he can create new values ($\mathsf{DY2}^{\mathsf{XP}}$), that he will learn everything sent on the ether ($\mathsf{DY3}^{\mathsf{XP}}$), that he can construct new composite values from known values using encryption and blinding ($\mathsf{DY4}^{\mathsf{XP}}$), that he can decrypt using known keys and unblind using known blinding factors ($\mathsf{DY5}^{\mathsf{XP}}$) and that he may forge new communication ($\mathsf{DY6}^{\mathsf{XP}}$).

Note that the rule for input ($\mathsf{DY3}^{\mathsf{XP}}$) and the rule for output ($\mathsf{DY6}^{\mathsf{XP}}$) of the attacker combined, will result in κ always equals $\rho(\mathsf{x}_\bullet)$. Hence κ could just be

$\rho, \kappa \models \bullet$ iff

$$\mathcal{N} \subseteq \rho(\mathsf{x}_\bullet) \wedge \qquad (\mathsf{DY1}^{XP})$$

$$\{\mathsf{n}^{\varepsilon}_\bullet, \mathsf{n}^{+}_\bullet, \mathsf{n}^{-}_\bullet\} \subseteq \rho(\mathsf{x}_\bullet) \wedge \qquad (\mathsf{DY2}^{XP})$$

$$\kappa \subseteq \rho(\mathsf{x}_\bullet) \wedge \qquad (\mathsf{DY3}^{XP})$$

$$\begin{array}{l} \forall k \geq 1 : \forall v_1 \in \rho(\mathsf{x}_\bullet), \cdots, \forall v_k \in \rho(\mathsf{x}_\bullet) : \\ \quad \mathsf{T}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \wedge \\ \forall v_0 \in \rho(\mathsf{x}_\bullet), \forall v_1 \in \rho(\mathsf{x}_\bullet) : \\ \quad \mathsf{H}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge \\ \quad \mathsf{E}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge \\ \quad \mathsf{P}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge \\ \quad \mathsf{B}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge \\ \quad \mathsf{S}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge \end{array} \qquad (\mathsf{DY4}^{XP})$$

$$\begin{array}{l} \forall v \in \rho(\mathsf{x}_\bullet) : \\ \quad (\forall k \geq 1 : \forall v_1, \cdots, v_k : \\ \qquad (v = \mathsf{T}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \Rightarrow \wedge_{i=1}^{k} v_i \in \rho(\mathsf{x}_\bullet)) \wedge \\ \quad \forall v_0, v_1 : \\ \qquad ((v = \mathsf{E}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge v_0 \in \rho(\mathsf{x}_\bullet)) \vee \\ \qquad (v = \mathsf{P}_{m^+}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge m^- \in \rho(\mathsf{x}_\bullet)) \vee \\ \qquad (v = \mathsf{B}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet) \wedge v_0 \in \rho(\mathsf{x}_\bullet)) \vee \\ \qquad v = \mathsf{S}_{v_0}(v_1) \in \rho(\mathsf{x}_\bullet)) \\ \qquad\quad \Rightarrow v_1 \in \rho(\mathsf{x}_\bullet)) \wedge \\ \quad \forall v_0, v_1, v' : \\ \qquad ((v = \mathsf{S}_{v'}(\mathsf{B}_{v_0}(v_1)) \in \rho(\mathsf{x}_\bullet) \wedge v_0 \in \rho(\mathsf{x}_\bullet)) \\ \qquad\quad \Rightarrow \mathsf{S}_{v'}(v_1) \in \rho(\mathsf{x}_\bullet))) \wedge \end{array} \qquad (\mathsf{DY5}^{XP})$$

$$\rho(\mathsf{x}_\bullet) \subseteq \kappa \qquad (\mathsf{DY6}^{XP})$$

Table 12.5: Analysis of the attacker; $\rho, \kappa \models_{\Gamma,\mathcal{N}} \bullet$.

used as the attackers knowledge, easing computations. However, we have chosen to keep $\mathsf{x}_\bullet$ in the description in order to explicitly represent his knowledge.

Chapter 13

Discussion of Part II

In the previous chapters, we have presented a new calculus for modelling protocols; LySA^{xp}, and a corresponding analysis. The aim of the calculus is to specify protocols in a clear and intuitive way, while still providing a sound and accurate analysis.

That LySA^{xp} specifications of protocols are readable and intuitive is exemplified by the optimised encoding of the FOO92 protocol in Table 11.3.

Additionally we have provided an analysis of LySA^{xp} and proved that this analysis is sound; ie. that the analysis components captures the entire behavior of the process. The accuracy; that is how few false-positives the analysis produces, has not been addressed previously and hence we shall discuss it in this chapter.

We have also described how to specify the attacker in LySA^{xp}, and the soundness of this attacker can be proven analogous to the proof from LySA in Chapter 5.

Furthermore we shall comment on a special kind of type-flaw attacks which could not be caught by the analysis of LySA, but will be by the analysis of LySA^{xp}.

13.1 Accuracy of the Analysis

The analysis of LySA$^{\text{XP}}$ uses the same restriction technique as LySA to make it efficiently computable; ie. canonical names and disciplined α-equivalence. However, the analysis of LySA$^{\text{XP}}$ differs from LySA on another point; the analysis of pattern matchings. The analysis enforces, that all possible variable bindings of the pattern matching are in ρ, before the pattern matching is performed. This may result in an over-approximation; eg. if the value $v = \mathsf{T}(A, B)$ is matched against the pattern $p = \mathsf{T}(\_\%x, C)$ the analysis $\rho \models v \sim p$ would update ρ with the variable binding of x to A, before the pattern matching $\rho \models v \triangleright p$ would fail, thus resulting in a variable binding that does not follow the semantics.

Whether over-approximations of this type will render the analysis unusable must be answered empirically. We can however describe how to extend the analysis, such that the accuracy is increased. We do this by extending the analysis with an additional abstract environment for variable bindings; σ. The idea is then, that given a value v, which should be matched against a pattern p, the analysis should first update σ with all possible variable bindings in the matching and then use σ to pattern match. Now only if the pattern matching using σ was successful, should the analysis update ρ with possible variable bindings, and then perform the pattern matching using ρ. This yields the following new analysis for input and match processes, given that $\forall x : \rho(x) \subseteq \sigma(x)$:

$$
\begin{array}{lll}
(\mathsf{AIn2^{XP}}) & \sigma, \rho, \kappa \models (p).P \quad \text{iff} & \\
 & \quad \forall v \in \kappa : \sigma \models v \sim p \wedge (& \\
 & \quad\quad \sigma \models v \triangleright p \Rightarrow (\rho \models v \sim p \wedge (& \\
 & \quad\quad\quad \rho \models v \triangleright p \Rightarrow \rho, \kappa \models P)) &
\end{array}
$$

$$
\begin{array}{lll}
(\mathsf{AMat2^{XP}}) & \sigma, \rho, \kappa \models \mathsf{match}\ t\ \mathsf{as}\ p.\, P \quad \text{iff} & \\
 & \quad \forall v \in \sigma[\![t]\!] : \sigma \models v \sim p \wedge (& \\
 & \quad\quad \sigma \models v \triangleright p \Rightarrow v \in \rho[\![t]\!] \Rightarrow (& \\
 & \quad\quad\quad \rho \models v \sim p \wedge (\rho \models v \triangleright p \Rightarrow \rho, \kappa \models P)) &
\end{array}
$$

which should drastically increase the accuracy of ρ. The only reason that we did not use this analysis in the first place, is that it requires extra computations, and should therefore only be employed if the analysis presented in Chapter 12 empirically proves not to be sufficient.

13.2 The Tuple Type-Flaw Attack

Originally the tuple construct was added to ease the analysis. However, when considering the effect of this change, it becomes clear that it also increases the

strength of the analysis. Consider the following scenario:

$$
\begin{array}{lll}
P ::= & (\nu^{\pm}\ K_S)\,(& \\
 & (\mathsf{S}_{K_S^+}(\mathsf{T}(\_,\_\%x))\,).\ & /*\,Client\,*/ \\
 & \vdots & \\
 \mid & (\_\%y).\ & /*\,Server\,*/ \\
 & \langle \mathsf{S}_{K_S^-}(y)\,\rangle.\mathsf{0} & \\
 &) &
\end{array}
$$

Where we have the two principals; the server and the client. Now the server accepts any message and returns it signed whereas at some point the user receives a message and ensures that this is a tuple signed by the server, before binding the second element of the tuple to x.

In LySA$^{\mathsf{xp}}$ the attacker could get any tuple signed by the server, and thus get them accepted by the client. If we wanted to find this attack in LySA we would have to extend the specification, such that the server also signed sequences of length two; ie. we would have to know the attack in advance. In essence this means that LySA$^{\mathsf{xp}}$ can find type-flaw attacks that LySA cannot; namely attacks where tuples are used as singular values.

Chapter 14

Conclusion

14.1 Related Work

In previous work [7, 8, 19] static program analysis has proved to be a simple and effective approach for validating confidentiality and authentication properties of key exchange protocols. In this thesis we have successfully used the very same approach for validating the somewhat different kind of security properties that apply for electronic voting protocols; namely verifiability, accuracy, democracy and fairness. We have studied one of the first voting protocols presented in the literature; the FOO92 protocol [18], and applied the very same framework to two newer voting protocols; Sensus [14] and E-Vox [20].

Automated validation of the security properties in a voting protocol was to our knowledge not presented in the literature at the beginning of our thesis. However during our thesis study, Kremer and Ryan published their work on the FOO92 protocol in [24]. Here they verify the properties fairness and the first part of democracy, by formalising the protocol in the applied π-calculus in order to use automated analysis with the tool ProVerif [6]. An additional property, privacy, is partly proved by hand using equivalence theory; that neither election authorities nor anyone else can link any vote to the voter who cast it.

Kremer and Ryan used the same definition of security properties for voting

protocols as the one we presented in Section 1.3. However, as their analysis was limited in several ways; eg. they could not distinguish the votes, they were not able to validate the remaining security properties.

The validation of the security properties of the Sensus and the E-vox protocols has in previous literature only relied on informal proofs [18, 14, 20].

14.2 Perspectives

The strategy for validating protocols is expressed by our framework in Figure 14.1 and we have made two approaches of this framework, described in each of the two parts in the thesis.

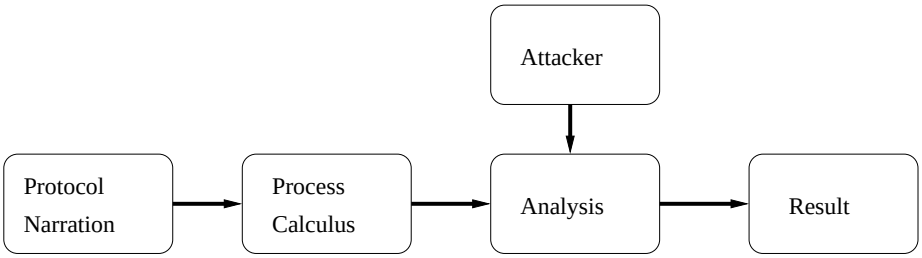


Figure 14.1: Framework

The first approach, as described in Part I, used the already existing process calculus LySA, which was extended with a blinding construct for the purpose. This extension is described thoroughly in Chapter 2, Chapter 4 and Chapter 5. Additionally the analysis of the extended LySA was implemented as described in Chapter 6, such that it could be applied to the voting protocols of choice.

Using this framework, our initial analysis results for verifiability and accuracy pinpoints flaws in the FOO92 and the Sensus protocols: We have identified a denial of service attack which could force the counter to repeatedly disqualify the voting process and we have identified a flaw which allows the attacker to forge the publishing of votes. However for both of these protocols we proposed an amendment, and the four security properties was subsequently validated for the amended protocol.

The E-Vox protocol had a bit different design, and the above mentioned flaws are not present in this protocol. However because of the protocol design, E-Vox

cannot satisfy fairness.

During the work behind Part I, the development of a new framework was motivated by the difficulties we experienced with specifying protocols in LySA. Therefore we introduce the LySA^{xp}-calculus in Part II; both syntax and semantics, and we proceed by presenting an analysis which we prove sound with respect to the semantics. That LySA^{xp} has a more intuitive syntax than LySA should be obvious from the FOO92 specification we presented in Chapter 3. However, further work on the LySA^{xp}-framework is required, before it can be applied on protocols.

First of all LySA^{xp} does not incorporate annotations yet. However, these can be added analogous to the annotations of LySA, and since they have no semantic consequence, the corresponding extension of the analysis is trivial.

The implementation of LySA^{xp} is a more interesting subject. As mentioned in Chapter 9; one of the motivations behind LySA^{xp} is an article by Buchholtz et al. [10], where they design a calculus with properties similar to LySA^{xp}. They also propose an analysis of their calculus but experience has shown that this analysis cannot be implemented easily. Hence we have taken an entirely different approach to the analysis; in particular we have divided the analysis of variable bindings and pattern matching up into two distinct analyses. The resulting analysis produces constraints quite similar to the analysis of LySA, and therefore we believe that this analysis could be implemented using the same technique as we presented in the implementation of LySA in Chapter 6.

Another, perhaps more interesting approach, would be trying to use another solver, such that some of the implementation steps would not be needed. Recently a variant of the Succinct Solver has been presented, which work with tree grammars, and hence does not require the transformation from infinite to finite. It should be interesting to see, if this solver will ease the implementation of LySA^{xp} .

14.3 Recapitulation

The contribution of this thesis is to the area of automated analysis of networking systems. More specifically, this thesis presents a framework for automatically validating the security properties of voting protocols. Our analysis covers four of the five security properties for voting protocols, and as the analysis calculates an over-approximation of the possible behavior of the protocols, we can be certain that the validation is sound.

We have used this framework to validate three voting protocols; the FOO92 voting protocol, one of the most established voting protocols in the literature, the Sensus voting protocol and the E-Vox voting protocol. Of these have, to the best of our knowledge, only the FOO92 protocol been validated previously in the literature, and that was only for some of the security properties. Thus this thesis also contributes with validation of protocols previously not validated.

Two different approaches to our framework has been studied; the LySA-calculus with corresponding analysis and tool was extended in order for us to analyse the voting protocols. The LySA^{xp}-calculus and a corresponding analysis was motivated by the complexity of modelling protocols in LySA, and we have seen that LySA^{xp} provides more intuitive protocol specifications. We have additionally proven the analysis to be sound with respect to the semantics, thus an interesting future project is to implement the analysis for LySA^{xp}.

Appendix A

LySA

A.1 Operational semantics for LySA with blinding

(Com) $\langle V_1, \ldots, V_k \rangle.P_1 \mid (V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k).P_2 \rightarrow P_1 \mid P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$

(SDec) $\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k\}_{V_0}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$

(ADec) $\mathsf{decrypt}\ \{\!|V_1, \ldots, V_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k|\!\}_{m^-}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$

(ASig) $\mathsf{decrypt}\ \{\!|V_1, \ldots, V_k|\!\}_{m^-}\ \mathsf{as}\ \{\!|V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k|\!\}_{m^+}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$

(UBli1) $\mathsf{unblind}\ [V_1, \ldots, V_k]_{V_0}\ \mathsf{as}\ [V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k]_{V_0}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$

(UBli2) $\mathsf{unblind}\ \{\!|[\![V_1, \ldots, V_k]\!]_{V_0}|\!\}_{m^-}\ \mathsf{as}\ [;\, x]_{V_0}\ \mathsf{in}\ P \rightarrow P[x \overset{\alpha}{\mapsto} \{\!|V_1, \ldots, V_k|\!\}_{m^-}]$

(New) $\dfrac{P \rightarrow P'}{(\nu\, n)\, P \rightarrow (\nu\, n)\, P'}$ (ANew) $\dfrac{P \rightarrow P'}{(\nu_{\pm}\, m)\, P \rightarrow (\nu_{\pm}\, m)\, P'}$

(Par) $\dfrac{P_1 \rightarrow P_1'}{P_1 \mid P_2 \rightarrow P_1' \mid P_2}$ (Congr) $\dfrac{P \equiv P'' \quad P'' \rightarrow P''' \quad P''' \equiv P'}{P \rightarrow P'}$

Table A.1: Reduction relation for processes; $P \rightarrow P'$.

$$
\begin{array}{ll}
P \equiv P & P \mid \mathsf{0} \equiv P \\
P_1 \equiv P_2 \Rightarrow P_2 \equiv P_1 & (\nu\, n)\, \mathsf{0} \equiv \mathsf{0} \\
P_1 \equiv P_2 \wedge P_2 \equiv P_3 \Rightarrow P_1 \equiv P_3 & (\nu_{\pm}\, m)\, \mathsf{0} \equiv \mathsf{0} \\
P_1 \equiv P_2 \text{ if } P_1 \text{ and } P_2 \text{ are } \alpha\text{-equivalent} & (\nu\, n_1)\, (\nu\, n_2)\, P \equiv (\nu\, n_2)\, (\nu\, n_1)\, P \\
P_1 \equiv P_2 \Rightarrow P_1 \mid P_3 \equiv P_2 \mid P_3 & (\nu_{\pm}\, m)\, (\nu\, n)\, P \equiv (\nu\, n)\, (\nu_{\pm}\, m)\, P \\
P_1 \mid P_2 \equiv P_2 \mid P_1 & (\nu_{\pm}\, m_1)\, (\nu_{\pm}\, m_2)\, P \equiv (\nu_{\pm}\, m_2)\, (\nu_{\pm}\, n_1)\, P \\
(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3) & P_1 \equiv P_2 \Rightarrow (\nu\, n)\, P_1 \equiv (\nu\, n)\, P_2 \\
!P \equiv P \mid !P & P_1 \equiv P_2 \Rightarrow (\nu_{\pm}\, m)\, P_1 \equiv (\nu_{\pm}\, m)\, P_2 \\
P_1 \equiv P_2 \Rightarrow !P_1 \equiv !P_2 & \\
(\nu\, n)\, (P_1 \mid P_2) \equiv P_1 \mid (\nu\, n)\, P_2 \quad \text{if } n \notin \mathsf{fn}(P_1) & \\
(\nu_{\pm}\, m)\, (P_1 \mid P_2) \equiv P_1 \mid (\nu_{\pm}\, m)\, P_2 \quad \text{if } m^+, m^- \notin \mathsf{fn}(P_1) &
\end{array}
$$

Table A.2: Structural congruence for processes; $P \equiv P'$.

$$
\begin{array}{ll}
P \overset{\alpha}{\equiv} P & \\
P_1 \overset{\alpha}{\equiv} P_2 \Rightarrow P_2 \overset{\alpha}{\equiv} P_1 & \\
P_1 \overset{\alpha}{\equiv} P_2 \wedge P_2 \overset{\alpha}{\equiv} P_3 \Rightarrow P_1 \overset{\alpha}{\equiv} P_3 & \\
 & \\
(\nu\, n_1)\, P \overset{\alpha}{\equiv} (\nu\, n_2)\, (P[n_1 \mapsto n_2]) & \text{if } n_2 \notin \mathsf{fn}(P) \\
(\nu_{\pm}\, m_1)\, P \overset{\alpha}{\equiv} (\nu_{\pm}\, m_2)\, (P[m_1^+ \mapsto m_2^+, m_1^- \mapsto m_2^-]) & \text{if } m_1^+, m_1^-, m_2^+, m_2^- \notin \mathsf{fn}(P)
\end{array}
$$

Table A.3: α-equivalence; $P \overset{\alpha}{\equiv} P'$.

$$
\begin{array}{lcl}
\mathsf{fn}(n) & \stackrel{\text{def}}{=} & \{n\} \\
\mathsf{fn}(m^+) & \stackrel{\text{def}}{=} & \{m^+\} \\
\mathsf{fn}(m^-) & \stackrel{\text{def}}{=} & \{m^-\} \\
\mathsf{fn}(x) & \stackrel{\text{def}}{=} & \emptyset \\
\mathsf{fn}(\{E_1, \ldots, E_k\}_{E_0}) & \stackrel{\text{def}}{=} & \mathsf{fn}(E_0) \cup \ldots \cup \mathsf{fn}(E_k) \\
\mathsf{fn}(\{\!|E_1, \ldots, E_k|\!\}_{E_0}) & \stackrel{\text{def}}{=} & \mathsf{fn}(E_0) \cup \ldots \cup \mathsf{fn}(E_k) \\
\mathsf{fn}([E_1, \ldots, E_k]_{E_0}) & \stackrel{\text{def}}{=} & \mathsf{fn}(E_0) \cup \ldots \cup \mathsf{fn}(E_k)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{fn}(\langle E_1, \ldots, E_k \rangle.P) & \stackrel{\text{def}}{=} & \mathsf{fn}(E_1) \cup \ldots \cup \mathsf{fn}(E_k) \cup \mathsf{fn}(P) \\
\mathsf{fn}((E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k).P) & \stackrel{\text{def}}{=} & \mathsf{fn}(E_1) \cup \ldots \cup \mathsf{fn}(E_j) \cup \mathsf{fn}(P) \\
\mathsf{fn}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P) & & \\
 & \stackrel{\text{def}}{=} & \mathsf{fn}(E) \cup \mathsf{fn}(E_0) \cup \ldots \cup \mathsf{fn}(E_j) \cup \mathsf{fn}(P) \\
\mathsf{fn}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k|\!\}_{E_0}\ \mathsf{in}\ P) & & \\
 & \stackrel{\text{def}}{=} & \mathsf{fn}(E) \cup \mathsf{fn}(E_0) \cup \ldots \cup \mathsf{fn}(E_j) \cup \mathsf{fn}(P) \\
\mathsf{fn}(\mathsf{unblind}\ E\ \mathsf{as}\ [E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k]_{E_0}\ \mathsf{in}\ P) & & \\
 & \stackrel{\text{def}}{=} & \mathsf{fn}(E) \cup \mathsf{fn}(E_0) \cup \ldots \cup \mathsf{fn}(E_j) \cup \mathsf{fn}(P) \\
\mathsf{fn}((\nu\, n)\, P) & \stackrel{\text{def}}{=} & \mathsf{fn}(P) \backslash \{n\} \\
\mathsf{fn}((\nu_{\pm}\, m)\, P) & \stackrel{\text{def}}{=} & \mathsf{fn}(P) \backslash \{m^+, m^-\} \\
\mathsf{fn}(P_1 \mid P_2) & \stackrel{\text{def}}{=} & \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2) \\
\mathsf{fn}(!P) & \stackrel{\text{def}}{=} & \mathsf{fn}(P) \\
\mathsf{fn}(0) & \stackrel{\text{def}}{=} & \emptyset
\end{array}
$$

Table A.4: Free names; $\mathsf{fn}(P)$.

A.2 Control flow analysis of LySA with blinding

$$
\begin{array}{lll}
\text{(AN)} & \rho \models n : \vartheta & \text{iff} \quad \lfloor n \rfloor \in \vartheta \\
\text{(ANp)} & \rho \models m^{+} : \vartheta & \text{iff} \quad \lfloor m^{+} \rfloor \in \vartheta \\
\text{(ANm)} & \rho \models m^{-} : \vartheta & \text{iff} \quad \lfloor m^{-} \rfloor \in \vartheta \\
\text{(AVar)} & \rho \models x : \vartheta & \text{iff} \quad \rho(\lfloor x \rfloor) \subseteq \vartheta \\
\text{(ASEnc)} & \rho \models \{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}] : \vartheta & \\
 & \quad \text{iff} \quad \wedge_{i=0}^{k} \rho \models E_i : \vartheta_i \wedge & \\
 & \qquad \forall\, V_0 \in \vartheta_0 \ldots V_k \in \vartheta_k : \{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta & \\
\text{(AAEnc)} & \rho \models \{\!|E_1, \cdots, E_k|\!\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}] : \vartheta & \\
 & \quad \text{iff} \quad \wedge_{i=0}^{k} \rho \models E_i : \vartheta_i \wedge & \\
 & \qquad \forall\, V_0 \in \vartheta_0 \ldots V_k \in \vartheta_k : \{\!|V_1, \cdots, V_k|\!\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta & \\
\text{(ABli)} & \rho \models [\![E_1, \cdots, E_k]\!]^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}] : \vartheta & \\
 & \quad \text{iff} \quad \wedge_{i=0}^{k} \rho \models E_i : \vartheta_i \wedge & \\
 & \qquad \forall\, V_0 \in \vartheta_0 \ldots V_k \in \vartheta_k : [\![V_1, \cdots, V_k]\!]^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}] \in \vartheta &
\end{array}
$$

Table A.5: Analysis of LySA terms; $\rho \models E : \vartheta$.

(AOut) $\rho, \kappa, \psi \models \langle E_1, \cdots, E_k \rangle.P$
iff $\wedge_{i=1}^{k} \rho \models E_i : \vartheta_i \wedge$
$\forall V_1 \in \vartheta_1 \ldots V_k \in \vartheta_k : V_1 \ldots V_k \in \kappa \wedge$
$\rho, \kappa, \psi \models P$

(AInp) $\rho, \kappa, \psi \models (E_1, \cdots, E_j; x_{j+1}, \cdots, x_k).P$
iff $\wedge_{i=1}^{j} \rho \models E_i : \vartheta_i \wedge$
$\forall V_1 \ldots V_k \in \kappa : \wedge_{i=1}^{j} V_i \in \vartheta_i \Rightarrow$
$\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P$

(ASDec) $\rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}^{\ell}_{E_0}[\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge$
$\forall \{V_1, \cdots, V_k\}^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'] \in \vartheta : \wedge_{i=0}^{j} V_i \in \vartheta_i \Rightarrow$
$\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi)$

(AADec) $\rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \cdots, E_j; x_{j+1}, \cdots, x_k|\!\}^{\ell}_{E_0}[\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge$
$\forall \{\!|V_1, \cdots, V_k|\!\}^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'] \in \vartheta :$
$\forall V_0' \in \vartheta_0 : \forall (\lfloor m^+ \rfloor, \lfloor m^- \rfloor) :$
$\{V_0, V_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge \wedge_{i=0}^{j} V_i \in \vartheta_i \Rightarrow$
$\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi)$

(AUBli) $\rho, \kappa, \psi \models \mathsf{unblind}\ E\ \mathsf{as}\ [\![E_1, \cdots, E_j; x_{j+1}, \cdots, x_k]\!]^{\ell}_{E_0}[\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P$
iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge$
$(\forall [\![V_1, \cdots, V_k]\!]^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'] \in \vartheta : \wedge_{i=0}^{j} V_i \in \vartheta_i \Rightarrow$
$\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$
$\wedge$
$(\forall \{\!|[\![V_1, \cdots, V_{k'}]\!]^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}']|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\ \mathcal{L}^{sig}] \in \vartheta :$
$j = 0 \wedge k = 1 \wedge V_0 \in \vartheta_0 \Rightarrow$
$\{\!|V_1, \cdots, V_{k'}|\!\}^{\ell^{sig}}_{V_0^{sig}}[\mathsf{dest}\ \mathcal{L}^{sig}] \in \rho(\lfloor x_1 \rfloor) \wedge$
$\rho, \kappa, \psi \models P \wedge$
$(\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L} \Rightarrow (\ell', \ell) \in \psi))$

(ANew) $\rho, \kappa, \psi \models (\nu\, n)\, P$ iff $\rho, \kappa, \psi \models P$

(AANew) $\rho, \kappa, \psi \models (\nu_{\pm}\, m)\, P$ iff $\rho, \kappa, \psi \models P$

(ARep) $\rho, \kappa, \psi \models !P$ iff $\rho, \kappa, \psi \models P$

(APar) $\rho, \kappa, \psi \models P_1 \mid P_2$ iff $\rho, \kappa, \psi \models P_1 \wedge \rho, \kappa, \psi \models P_2$

(ANil) $\rho, \kappa, \psi \models 0$ iff true

Table A.6: Analysis of LySA processes; $\rho, \kappa, \psi \models P$.

A.3 Extending the LySaTool

The following describes the changes we did to the LySATool implementation.

- made an optimised version of the set implementation in set.sml, the original one was very inefficient;
- added the following rule for unblinding to lysalexer.lex

```
<INITIAL> unblind => (Tokens.UNBLIND(!pos,!pos));
```

- added a rule for blinding to term in lysagrammar.grm:

```
| LSBRACKET BAR termlist BAR RSBRACKET COLON term dest
    (let val (c,ol,ls) = dest
     in MLLysa.BLI(termlist,term,c,ol,ls,newlab()) end)
```

- added a rule for unblinding to proc in lysagrammar.grm:

```
| UNBLIND term AS
  LSBRACKET BAR termlist SEMICOLON varlist BAR RSBRACKET COLON term
  orig IN proc
    (let val (c,ol,ls) = orig
     in MLLysa.UBLI(term1,termlist,varlist,term2,c,ol,ls,proc)
     end)
```

- changed lysagrammar.grm so all variables also get unique labels, the original implementation was not completely clear at this point as some optimisations had been added here;
- added several lines to mllysa.sml all marked by (* A *), these are the changes to the syntax of LySa;
- added several lines to lysaasciiio.sml, lysahtmlio.sml and lysalatexio.sml all marked by (* A *), these are the files used for converting the internal representation to different external representations;
- added e-lysa.sty, a latex style for the extended LySa;
- added several lines to analysis2.sml all marked by (* A *), this is the actual extension of the analysis as presented in Chapter 4.

Appendix B

LySA^{XP}

B.1 Example of the analysis

This appendix shows the full calculations of applying the analysis to the protocol presented in Example 12.1.

$$
\begin{array}{lll}
P ::= & (\nu^{\varepsilon}\ B)\,(\nu^{\varepsilon}\ msg) & /*\,Client\,*/ \\
& \langle \mathsf{B}_B(msg)\,\rangle. & \\
& (\mathsf{B}_B(\_\%x)\,). & \\
& \langle x\rangle.\mathsf{0} & \\
\mid & (\nu^{\pm}\ K_S) & /*\,Server\,*/ \\
& (\_\%y). & \\
& \langle \mathsf{S}_{K_S^-}(y)\,\rangle.\mathsf{0} &
\end{array}
$$

The first step is to recursively apply the analysis of processes:

$$
\begin{array}{lll}
\rho,\kappa \models P & \text{iff} & \rho,\kappa \models (\nu^{\varepsilon}\ B)\,(\nu^{\varepsilon}\ msg)\,\langle \mathsf{B}_B(msg)\,\rangle.(\mathsf{B}_B(\_\%x)\,).\langle x\rangle.\mathsf{0} \\
& & \wedge \\
& & \rho,\kappa \models (\nu^{\pm}\ K_S)\,(\_\%y).\langle \mathsf{S}_{K_S^-}(y)\,\rangle.\mathsf{0} \\
& & \\
& \text{iff} & \rho,\kappa \models \langle \mathsf{B}_B(msg)\,\rangle.(\mathsf{B}_B(\_\%x)\,).\langle x\rangle.\mathsf{0} \\
& & \wedge \\
& & \rho,\kappa \models (\_\%y).\langle \mathsf{S}_{K_S^-}(y)\,\rangle.\mathsf{0} \\
& & \\
& \text{iff} & (\rho[\mathsf{B}_B(msg)\,] \subseteq \kappa \ \wedge\ \rho,\kappa \models (\mathsf{B}_B(\_\%x)\,).\langle x\rangle.\mathsf{0}) \\
& & \wedge \\
& & (\forall v \in \kappa : \rho \models v \sim \_\%y \ \wedge\ (\\
& & \quad \rho \models v \triangleright \_\%y \Rightarrow \rho,\kappa \models \langle \mathsf{S}_{K_S^-}(y)\,\rangle.\mathsf{0})) \\
& & \\
& \text{iff} & (\rho[\mathsf{B}_B(msg)\,] \subseteq \kappa \ \wedge\ (\\
& & \quad \forall v \in \kappa : \rho \models v \sim \mathsf{B}_B(\_\%x) \ \wedge\ (\\
& & \quad\quad \rho \models v \triangleright \mathsf{B}_B(\_\%x) \ \Rightarrow\ (\\
& & \quad\quad \rho[x] \subseteq \kappa \ \wedge\ \rho,\kappa \models \mathsf{0})))) \\
& & \wedge \\
& & (\forall v \in \kappa : \rho \models v \sim \_\%y \ \wedge\ (\\
& & \quad \rho \models v \triangleright \_\%y \Rightarrow (\\
& & \quad\quad \rho[\mathsf{S}_{K_S^-}(y)\,] \subseteq \kappa \ \wedge\ \rho,\kappa \models \mathsf{0}))) \\
& & \\
& \text{iff} & (\rho[\mathsf{B}_B(msg)\,] \subseteq \kappa \ \wedge\ (\\
& & \quad \forall v \in \kappa : \rho \models v \sim \mathsf{B}_B(\_\%x) \ \wedge\ (\\
& & \quad\quad \rho \models v \triangleright \mathsf{B}_B(\_\%x) \ \Rightarrow\ (\\
& & \quad\quad \rho[x] \subseteq \kappa \ \wedge\ \mathsf{true})))) \\
& & \wedge \\
& & (\forall v \in \kappa : \rho \models v \sim \_\%y \ \wedge\ (\\
& & \quad \rho \models v \triangleright \_\%y \Rightarrow (\\
& & \quad\quad \rho[\mathsf{S}_{K_S^-}(y)\,] \subseteq \kappa \ \wedge\ \mathsf{true})))
\end{array}
$$

We then apply the evaluation of terms, analysis of bindings and pattern matching:

$\rho, \kappa \models P$

iff $(\rho[\![\mathsf{B}_B(msg)]\!] \subseteq \kappa \wedge ($
$\forall v \in \kappa : \rho \models v \sim \mathsf{B}_B(\_\%x) \wedge ($
$\rho \models v \triangleright \mathsf{B}_B(\_\%x) \Rightarrow ($
$\rho[\![x]\!] \subseteq \kappa \wedge \mathsf{true}))))$
$\wedge$
$(\forall v \in \kappa : \rho \models v \sim \_\%y \wedge ($
$\rho \models v \triangleright \_\%y \Rightarrow ($
$\rho[\![\mathsf{S}_{K_S^-}(y)]\!] \subseteq \kappa \wedge \mathsf{true})))$

iff $(\{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge ($
$\forall v \in \kappa : \rho \models v \sim \mathsf{B}_B(\_\%x) \wedge ($
$\rho \models v \triangleright \mathsf{B}_B(\_\%x) \Rightarrow ($
$\rho(\lfloor x \rfloor) \subseteq \kappa \wedge \mathsf{true}))))$
$\wedge$
$(\forall v \in \kappa : \rho \models v \sim \_\%y \wedge ($
$\rho \models v \triangleright \_\%y \Rightarrow ($
$\{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa \wedge \mathsf{true})))$

iff $(\{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge ($
$\forall v \in \kappa : ($
$(v = \mathsf{B}_{v_0}(v') \Rightarrow$
$(\rho \models v_0 \triangleright B \Rightarrow \rho \models v' \sim \_\%x)) \vee$
$(v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \Rightarrow$
$(\rho \models v_0 \triangleright B \Rightarrow \rho \models \mathsf{S}_{m^-}(v') \sim \_\%x))) \wedge ($
$\rho \models v \triangleright \mathsf{B}_B(\_\%x) \Rightarrow ($
$\rho(\lfloor x \rfloor) \subseteq \kappa \wedge \mathsf{true}))))$
$\wedge$
$(\forall v \in \kappa : \rho \models \_\%y \sim v \wedge ($
$\rho \models v \triangleright \_\%y \Rightarrow ($
$\{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa \wedge \mathsf{true})))$

iff $(\{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge ($
$\forall v \in \kappa : ($
$(v = \mathsf{B}_{v_0}(v') \Rightarrow$
$(\rho \models v_0 \triangleright B \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v' \in \rho(x))))) \vee$
$(v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \Rightarrow$
$(\rho \models v_0 \triangleright B \Rightarrow$
$(\mathsf{true} \wedge (\mathsf{true} \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x))))) \wedge ($
$\rho \models v \triangleright \mathsf{B}_B(\_\%x) \Rightarrow ($
$\rho(\lfloor x \rfloor) \subseteq \kappa \wedge \mathsf{true}))))$
$\wedge$
$(\forall v \in \kappa : (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v \in \rho(y))) \wedge ($
$\rho \models v \triangleright \_\%y \Rightarrow ($
$\{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa \wedge \mathsf{true})))$

$$
\begin{array}{ll}
\text{iff} & (\{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge (\\
& \quad \forall v \in \kappa : (\\
& \quad (v = \mathsf{B}_{v_0}(v') \Rightarrow \\
& \qquad (\rho \models v_0 \triangleright B \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v' \in \rho(x))))) \vee \\
& \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))) \Rightarrow \\
& \qquad (\rho \models v_0 \triangleright B \Rightarrow \\
& \qquad (\mathsf{true} \wedge (\mathsf{true} \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x))))))) \wedge (\\
& \qquad ((v = \mathsf{B}_{v_0}(v') \wedge \rho \models v_0 \triangleright B \wedge \rho \models v' \triangleright \_\%x) \vee \\
& \qquad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))) \wedge \rho \models v_0 \triangleright B \wedge \rho \models \mathsf{S}_{m^-}(v') \triangleright \_\%x)) \Rightarrow (\\
& \qquad\quad \rho(\lfloor x \rfloor) \subseteq \kappa \wedge \mathsf{true})))) \\
& \wedge \\
& (\forall v \in \kappa : (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v \in \rho(y))) \wedge (\\
& \quad \rho \models v \triangleright \_\%y \Rightarrow (\\
& \qquad \{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa \wedge \mathsf{true})))
\end{array}
$$

$$
\begin{array}{ll}
\text{iff} & (\{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge (\\
& \quad \forall v \in \kappa : (\\
& \quad (v = \mathsf{B}_{v_0}(v') \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v' \in \rho(x))))) \vee \\
& \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))) \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x))))))) \wedge (\\
& \qquad ((v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B \rfloor \wedge \mathsf{true}) \vee \\
& \qquad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))) \wedge v_0 = \lfloor B \rfloor \wedge \mathsf{true})) \Rightarrow (\\
& \qquad\quad \rho(\lfloor x \rfloor) \subseteq \kappa \wedge \mathsf{true})))) \\
& \wedge \\
& (\forall v \in \kappa : (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v \in \rho(y))) \wedge (\\
& \quad \mathsf{true} \Rightarrow (\\
& \qquad \{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa \wedge \mathsf{true})))
\end{array}
$$

By using simple logics we can reduce this rather complex constraint.

$$
\begin{array}{ll}
\rho, \kappa \models P & \\
\text{iff} & (\{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge (\\
& \quad \forall v \in \kappa : (\\
& \quad (v = \mathsf{B}_{v_0}(v') \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v' \in \rho(x))))) \vee \\
& \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))) \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow (\mathsf{true} \wedge (\mathsf{true} \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x))))))) \wedge (\\
& \qquad ((v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B \rfloor \wedge \mathsf{true}) \vee \\
& \qquad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v'))) \wedge v_0 = \lfloor B \rfloor \wedge \mathsf{true})) \Rightarrow (\\
& \qquad\quad \rho(\lfloor x \rfloor) \subseteq \kappa \wedge \mathsf{true})))) \\
& \wedge \\
& (\forall v \in \kappa : (\mathsf{true} \wedge (\mathsf{true} \Rightarrow v \in \rho(y))) \wedge (\\
& \quad \mathsf{true} \Rightarrow (\\
& \qquad \{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa \wedge \mathsf{true})))
\end{array}
$$

$$
\begin{array}{ll}
\text{iff} & \{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge (\\
& \quad \forall v \in \kappa : (\\
& \quad (v = \mathsf{B}_{v_0}(v') \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow v' \in \rho(x))) \vee \\
& \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x)))) \wedge (\\
& \qquad ((v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B \rfloor) \vee \\
& \qquad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \wedge v_0 = \lfloor B \rfloor)) \Rightarrow (\\
& \qquad\quad \rho(\lfloor x \rfloor) \subseteq \kappa))) \\
& \wedge \\
& (\forall v \in \kappa : v \in \rho(y)) \wedge (\\
& \quad \{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa))
\end{array}
$$

$$
\begin{array}{ll}
\text{iff} & \{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge \\
& \forall v \in \kappa : (\\
& \quad (v = \mathsf{B}_{v_0}(v') \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow v' \in \rho(x))) \vee \\
& \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \Rightarrow \\
& \qquad (v_0 = \lfloor B \rfloor \Rightarrow \mathsf{S}_{m^-}(v') \in \rho(x))) \wedge \\
& \quad (((v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B \rfloor) \vee \\
& \quad (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \wedge v_0 = \lfloor B \rfloor)) \Rightarrow (\\
& \qquad \rho(\lfloor x \rfloor) \subseteq \kappa))) \\
& \wedge \\
& (\kappa \subseteq \rho(y)) \wedge \\
& (\{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa)
\end{array}
$$

$$
\begin{array}{ll}
\text{iff} & \{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge \\
& \forall v \in \kappa : (\\
& \quad (((v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B \rfloor) \Rightarrow \\
& \qquad (v' \in \rho(x) \wedge \rho(\lfloor x \rfloor) \subseteq \kappa)) \vee \\
& \quad ((v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \wedge v_0 = \lfloor B \rfloor) \Rightarrow \\
& \qquad (\mathsf{S}_{m^-}(v') \in \rho(x) \wedge \rho(\lfloor x \rfloor) \subseteq \kappa)))) \\
& \wedge \\
& (\kappa \subseteq \rho(y)) \wedge \\
& (\{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa)
\end{array}
$$

$$
\begin{array}{ll}
\text{iff} & \{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \subseteq \kappa \wedge \\
& (\forall v \in \kappa : (v = \mathsf{B}_{v_0}(v') \wedge v_0 = \lfloor B \rfloor) \Rightarrow \\
& \qquad (v' \in \rho(x) \wedge \rho(\lfloor x \rfloor) \subseteq \kappa)) \wedge \\
& (\forall v \in \kappa : (v = \mathsf{S}_{m^-}(\mathsf{B}_{v_0}(v')) \wedge v_0 = \lfloor B \rfloor) \Rightarrow \\
& \qquad (\mathsf{S}_{m^-}(v') \in \kappa\rho(x) \wedge \rho(\lfloor x \rfloor) \subseteq \kappa)) \wedge \\
& (\kappa \subseteq \rho(y)) \wedge \\
& (\{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|\, v \in \rho(\lfloor y \rfloor)\} \subseteq \kappa)
\end{array}
$$

Which can easily be seen to be solved with the following smallest solution:

$$\begin{array}{lcl} \kappa & = & \{\mathsf{B}_{\lfloor B \rfloor}(\lfloor msg \rfloor)\} \cup \{\mathsf{S}_{\lfloor K_S^- \rfloor}(v) \,|v \in \kappa\} \cup \rho(x) \\ \rho(x) & = & \{\mathsf{S}_{\lfloor K_S^- \rfloor}(\lfloor msg \rfloor), \lfloor msg \rfloor\} \\ \rho(y) & = & \kappa \end{array}$$

Bibliography

[1] M. Abadi. Security protocols and specifications. In *FoSSaCS '99: Proceedings of the Second International Conference on Foundations of Software Science and Computation Structure, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99*, pages 1–13. Springer-Verlag, 1999.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–115. ACM Press, 2001.

[3] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols The Spi Calculus. Research Report 149, SRC - Systems Research Center, 1998.

[4] T. Asano, T. Matsumoto, and H. Imai. A study on some schemes for fair electronic secret voting. In *The Proceedings of the 1991 Symposium on Cryptography and Information Security*, SCIS91-12A, Feb. 1991. (in japanese).

[5] J. D. C. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, 1987.

[6] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW '01: Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, page 82, Washington, DC, USA, 2001. IEEE Computer Society.

[7] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Automatic Validation of Protocol Narration. In *CSFW*, pages 126–140, 2003.

[8] C. Bodei, M. Buchholtz, P. Degano, H. R. Nielson, and F. Nielson. Static Validation of Security Protocols. *Journal of Computer Security*, 2004.

[9] M. Buchholtz. Implementing Control Flow Analysis for Security Protocols. Technical Report WP6-IMM-I00-Int-003, DEGAS, 2004.

[10] M. Buchholtz, H. R. Nielson, and F. Nielson. A calculus for control flow analysis of security protocols. *Int. J. Inf. Sec.*, 2(3-4):145–167, 2004.

[11] D. Chaum. Blind Signatures for Untraceable Payments. *Advances in Cryptology Proceedings of Crypto 82*, pages 199–203, 1982.

[12] D. Chaum. Security without Identification : Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[13] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1997. http://www.grappa.univ-lille3.fr/tata.

[14] L. F. Cranor and R. K. Cytron. Design and Implementation of a Practical Security-Conscious Electronic Polling System. Research Report WUCS-96-02, Department of Computer Science, Washington University, 1996.

[15] http://www.cryptyc.org/, 2001. Webpage of Cryptyc.

[16] D. Dolev and A. Yao. On the Security of Public Key Protocols. *Proc. 22th IEEE Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[17] http://theory.lcs.mit.edu/~cis/voting/voting.html, 1997. DARPA contract DABT63-96-C-0018.

[18] A. Fujioka, T. Okamoto, and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. *Lecture Notes in Computer Science: Advances in Cryptology - AUSCRYPT '92*, 718:244–251, 1992.

[19] S. M. Hansen, J. Skriver, and H. R. Nielson. Using Static Analysis to Validate the SAML Single Sign-On Protocol. In *WITS '05: Proceedings of the 2005 workshop on Issues in the theory of security*, pages 27–40. ACM Press, 2005.

[20] M. A. Herschberg. Secure electronic voting over the world wide web. M.Sc. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1997.

[21] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. *Lecture Notes in Computer Science*, 1807:539+, 2000.

[22] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 26(1):100–106, 1983.

[23] J. Kilian and K. Sako. Receipt-free MIX-type voting scheme - a practical solution to the implementation of a voting booth. In *Proceedings of EUROCRYPT 1995*. Springer-Verlag, 1995.

[24] S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, Edinburgh, U.K., 2005. Springer.

[25] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[26] LySaTool, 2004.

[27] M. Maidl. Finding Replay Attacks In LySa. Invited talk, LySa workshop, Mar. 2005.

[28] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, fifth edition edition, 1999.

[29] M. Naor. Bit Commitment Using Pseudo-Randomness. In *CRYPTO*, pages 128–136, 1989.

[30] C. R. Nielsen, E. H. Andersen, and H. R. Nielson. Static validation of a voting protocol. In P. Degano and L. Viganó, editors, *Proc. ARSPA 2005, proceedings of the 2nd workshop on Automated Reasoning for Security Protocol Analysis*, pages 115–134. ENTCS 135(1), 2005.

[31] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[32] F. Nielson, H. R. Nielson, and R. R. Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002.

[33] F. Nielson, H. R. Nielson, and H. Seidl. A succinct solver for ALFP. *Nordic Journal of Computing*, 9:335–372, 2002.

[34] H. R. Nielson and F. Nielson. Flow logics for constraint based analysis. In *Proc. CC'98*, number 1383 in Lecture Notes in Computer Science, pages 109–127. Springer, 1998.

[35] H. R. Nielson and F. Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation: Complexity, Analysis, Transformation.*, volume 2566 of *Lecture Notes in Computer Science*, pages 223–244. Springer, 2002.

[36] Standard ML of New Jersey. http://www.smlnj.org/.

[37] P. Syverson. A Taxonomy of Replay Attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 187–191, 1994.