Using Static Analysis to Validate SAML Protocols

Master's Thesis by
Jakob Skriver &
Steffen M. Hansen

Instutute of Informatics and Mathematical Modelling
Technical University of Denmark
IMM-THESIS-2004-84
Kgs. Lyngby November 26, 2004

Abstract

Previous studies have successfully used static analysis to automatically validate security properties of classical protocols. In this thesis we show how the very same technique can be used to validate modern web-based protocols, in particular, we study the SAML Single Sign-On protocols.

The specifications of the protocols does not supply any security analysis. Different kinds of attacks are discussed and various recommendations on the security which must be used on each messages sent. We model the protocols using the process calculus LySa and using the static analysis tool LySa-tool we can analyse them. We are able to find flaws in the protocols even when following the recommendations of the specifications. The attacks found is also not discussed in the specifications.

To help us writing the large LySa processes needed for SAML Single Sign-On we extend the LySa process calculus with macro language. This enables us to specify transport layer protocols and the protocols above with little effort.

Keywords: Protocol Validation, LySa, Static Analysis, Authentication, Authentication Protocols.

Resumé

Tidligere studier har med stor succes brugt statisk analyse til automatisk at validere sikkerhedsegenskaber i klassiske protokoller. I denne afhandling viser vi hvordan de samme teknikker også kan bruges til at validere moderne web baserede protokoller. Vi vil mere specifikt behandle SAML Single Sign-On protokollerne.

Specifikationen af protokollerne indeholder ikke nogen sikkerhedsanalyse af protokollerne. Dog gennemgås forskellige former for angreb imod protokollerne samt forskellige anbefalinger af sikkerhed der skal bruges ved hver besked der sendes. Vi modellerer protokollerne ved brug af proceskalkulen LySa og analyserer dem ved hjælp af LySa-toolder bygger på statisk analyse. Vi finder flere fejl i protokollerne selv når vi følger anbefalingerne for sikkerhed. De fundne angreb er heller ikke beskrevet i specifikationerne for protokollerne.

For at hjælpe os med at skrive de store LySa processer der er nødvendige for SAML Single Sign-On tilføjer vi et macro sprog til LySa. Dette gør os i stand til lettere at specificere transport lags protokoller og de overliggende protokoller.

Nøgleord: Protokol validering, LySa, Statisk analyse, Autentifikation, Autentifikationsprotokoller.

Preface

This thesis describes and documents the M.Sc. Thesis Project of Steffen M. Hansen and Jakob Skriver. The project corresponds to 35 ECTS points and was carried out in the period from June to November 2004. The project was carried out at the Technical University of Denmark, Department of Informatics and Mathematical Modelling under the supervision of Professor Hanne Riis Nielson.

Parts of this thesis are to appear in the paper *Using static analysis to validate the SAML Single Sign-On protocol* [18] and are going to be presented on the Workshop on Issues in the Theory of Security (WITS'05) [43].

Acknowledgements. First of all we would like to thank Hanne Riis Nielson, for her great interest in our work, and for her many encouraging and helpful comments. Many helpful comments and ideas have come from members of the Programming Language Technology group at Informatics and Mathematical Modelling. In particular we would like to thank Mikael Buchholtz for discussing the topic of LySa with us and his great help on fine-tuning the implementation of the analysis. Also we have profited greatly of the very thorough comments from the reviewers at WITS [43]. They have pointed out many things which needed clarification.

Jakob Skriver
s991305

Steffen M. Hansen
s991471

Lyngby November 26, 2004

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

So far the application of formal methods in the analysis of cryptographic protocols has mainly been concerned with key distribution protocols used to establish secure communication between two principals; the Needham Schroeder shared key and public key protocols are famous example of protocols in this category [30]. In many cases automatic or semi-automatic tools have been constructed based on the theoretical developments and they have successfully pinpointed subtle bugs in protocols that have been considered secure for several years; a success story is Lowe's attack [22, 23] on the Needham Schroeder public key protocol which was discovered using the FDR model checker [16].

In this thesis we will describe and use a formal and automatic method to verify protocols used in modern web-scenarios. In particular we will study the SAML Single Sign-On protocol.

1.1 Cryptographic protocols

The purpose of cryptographic protocols is to establish and perform secure communication on an insecure network. Principals in a protocol use cryptographic functions and shared secrets[1] either to prove that they are in fact who they claim to be, or to transfer confidential data over the network (or both).

A message transfer such as principal A sending message M encrypted under the key K to principal B has the notation:

$$A \rightarrow B \;:\; \{M\}_K$$

Later in this thesis we will extend this notation. To be able to decrypt (and read) the message M, principal B must possess the key K. A protocol is formalised as a list of correct message transfers. To exemplify this, the following notation describes a variant of the Wide Mouthed Frog protocol (WMF) [12]:

$$\begin{array}{lll} 1. & A \rightarrow S: & A, B, \{K\}_{K_A} \\ 2. & S \rightarrow B: & A, \{K\}_{K_B} \\ 3. & A \rightarrow B: & \{m_1, ..., m_k\}_K \end{array}$$

This protocol aims to establish a shared session-key K between the two principals A and B, who prior to this protocol run have shared master keys K_A and

[1]Secret to all others than the intended principals

K_B with a trusted server S. The principal A initiates the protocol by sending the message $A, B\{K\}_{K_A}$ to the server S in step 1. Since S possesses the shared master key K_A, S is able to decrypt the message and recognise that A wants to engage secure communication with B using K as a session key. In step 2 S sends the message $A, \{K\}_{K_B}$ to B. B is able to decrypt the message and retrieve the session key K. In step 3 A is now able to send a secret message $\{m_1, ..., m_k\}_K$ to B using the session key K, since the key now is known to both A and B.

In this example only symmetric encryption has been used. Asymmetric encryption is also a very common technique in cryptographic protocols, this is carried out using a private/public key pair e.g. K^-/K^+. The private key is kept secret and the public key is common knowledge. Encrypting messages using a private key leaves the opportunity for all to decrypt it using the public key, whether as encrypting messages using the public key only allow for the principal possessing the private key to decrypt it. Description of asymmetric encryption is done in the following notation:

$$A \rightarrow B \ : \ \{|M|\}_{K^-}$$

This describes the scenario where principal A sends the message M encrypted using his private key K^-. If the principal B should be able to decrypt the message, the principal must possess the corresponding public key K^+.

It is important to note that since public keys usually are public in the sense that every principal is able to obtain it, asymmetric encryption itself does not assure, the message to be kept secret.

The global scenario

In the formalisation of the protocol in the previous Section, only the correct message transfers of the protocol are described, it is therefore important to be aware of the possibility of an attacker present on the network. A common way to model an attacker on a public accessed network, is to use the Dolev-Yao notion of a "hardest attacker" [14]. This model allows the attacker to perform the following operations:

- The attacker is able to intercept any message.
- The attacker can decrypt an encrypted message if and only if he knows the key. The attacker can encrypt messages using keys in his possession. The attacker cannot guess a key.
- The attacker can construct new messages.
- The attacker can send constructed or intercepted messages on the network.

In this scenario it is not possible to determine the sender of a message by looking at it. Since the attacker can intercept and replay any messages, nor the origin neither the destination of any message is certain, while an attacker is present.

If the WMF protocol from the previous Section is deployed on a network where an attacker is present, the following run of the protocol could occur:

$$\begin{array}{llcll}
1. & A & \rightarrow & M(S) & : \ A, B, \{K\}_{K_A} \\
1.' & M(A) & \rightarrow & S & : \ A, M, \{K\}_{K_A} \\
2. & S & \rightarrow & M(B) & : \ A, \{K\}_{K_M} \\
3. & A & \rightarrow & M(B) & : \ \{m_1, ..., m_k\}_K
\end{array}$$

$M(S)$ denotes the malicious attacker acting as S in this message transfer. The attacker intercepts the first message sent from A to the server. Since the attacker does not possess the key K_A he cannot decrypt the session key. The attacker now changes the intercepted message and replaces B with his own identifier M. Receiving this the server S believes that A wants to engage secure communication with the attacker M, and therefore the session key is sent to the attacker encrypted with a master key shared between the attacker and the server. Now the attacker is able to intercept and read messages sent from A to B encrypted under the session key K. Messages believed by A to be secret between A and B are in fact readable to the attacker.

Validating protocols

Doing a validation of a protocol, it is important to be aware of what properties of the protocol is validated. A protocol validated to be tolerant to denial of service attacks could very well be flawed with respect to replay attacks. The most common properties to consider when validating cryptographic protocols are:

Authenticity To be authenticated means to prove that you are in fact the one that you claim to be. Communicating over a protocol which offers authenticity means that you are communicating with the exact principal you believe to be communicating with.

Confidentiality A protocol that ensures confidentiality ensures that secret data is transfered in a way, such that only the intended receiver is able to read the data.

Integrity If data integrity is offered, the principals know that messages cannot be changed by any malicious user.

The task of validating these properties is very difficult, and many different approaches have been used over the last decades. The most recent research tend to formalise protocols in some simplified programming language, process calculus or logic description, and then use automatic tools to verify the properties for the simplified description of the protocol. The three main approaches in automatic verification are:

- Theorem proving methods use automatic tool to carry out proofs on a protocol described in a logic formalisation. This method is used in [6] to verify the SET protocol.
- State exploration methods [16], from model checking have been used to find flaws in protocols. This method explores each state in the protocol and reports if the protocol enters a state that violates the properties to be validated.
- Static analysis has also successfully detected errors in protocols [8]. Control flow analysis is used to do an over-approximation of the possible variable bindings and message transfers. Constructing reference monitor semantics it is possible to know whether the properties to be validated are violated.

The theorem proving method is often used on the classic small key distribution protocols, where the reasoning about the formalisation of the protocol into a logic description is relatively simple, and the assumption made prior to a run of the protocol. For more modern protocols, involving complex scenarios and big applications these reasonings become very hard to do.

The methods from model checking and static analysis are somewhat similar; in these two cases a reachability analysis is carried out. Confidentiality is interpreted as weather secret data reaches the attacker. Authentication is reachability in the sense that information should end up at the user from the intended provider of that information. Both methods have advantages and disadvantages. The model checking approach will always return a trace of the protocol that leads to the reported error, but it have to investigate all possible traces trough the protocol. As the length of the protocol to be analysed increases the number of different traces through a protocol raises significantly, and if an attacker is present, the number of states is infinite which makes it hard to use the method on full scale protocols. In static analysis it is possible to create an over-approximation of the components, without investigating all possible traces, this makes it feasible to create automatic tools for validations with the presence of an attacker. If an error is reported by the static analysis, the trace leading to the error is however not part of the result.

1.2 Strategy and Concepts

In this thesis we describe the method we have used, to analyse security properties of the SAML Single Sign-On protocol. The task is done in several steps:

1. Derive a model of the protocol and describe it in the LySa process calculus.
2. A static analysis of the LySa process is carried out. This analysis reveal potential breaches in the protocol.
3. The result of the static analysis is analysed.
4. If the analysis reveals attacks on the protocols, we shall suggest modifications to avoid these attacks.

The analyses of the protocols are carried out within the LySa framework [8]. LySa is a process calculus in the pi/spi calculus family [3, 27, 28] allowing communication protocols to be specified and annotated allowing for validation of authentication properties, in particular, destination/origin authentication properties. To be able to describe a modern scenario, we have extended the original LySa-calculus. This enables us to add different transport layers to the message transfers.

The technology of our analysis is based on static analysis [31] meaning that we conservatively construct an approximation of the behaviour of the protocol. In doing so we focus on (*i*) the communications that may take place over the network, (*ii*) the potential bindings of the variables occurring in the protocol and (*iii*) the potential violations of the destination/origin annotations of the protocol. The analysis is carried out in the presence of a Dolev-Yao attacker [14] meaning that any message sent on the network may be intercepted, any encryption with a key known to the attacker may be decrypted by him and furthermore the attacker may make use of all the information available to him to construct new encryptions and to send messages on the network. To be able to model encryption we use the notion of a *perfect encryption library*; after encryption a message can only be decrypted if the correct key is used.

The protocol we have chosen to analyse is the SAML Single Sign-On protocol. SAML stands for Security Assertion Markup Language and the Single Sign-On protocol is developed by the standardisation organisation OASIS [41] for secure exchange of information between online business partners, in particular for authentication of web-users to destination sites via already established authentications with source sites.

The SAML Single Sign-On protocol operates with three kinds of principals: users, destination sites and source sites. Users will (via a standard web-browser) communicate with a source site in order to be authenticated and thereby get access to a destination site offering some specific services; these services are only available to users that have been authenticated by one of the source sites. The actual formats of the messages exchanged are described using XML; they include a large number of elements, however, only a few are mandatory thereby leaving a lot of security decisions to the implementation. From the description above it should already be evident that the Single Sign-On protocol is not just *one* protocol but rather a framework for several protocols, a fact that is further stressed by the advice from [25]:

> *Before deployment, each combination of authentication, message integrity, and confidentiality mechanisms should be analysed for vulnerability in the context of the deployment environment.*

In this thesis we shall analyse several variants of the protocol.

One of the recommendations of the OASIS group is to use the Transport Layer Security (TLS) protocol as the basis for the communication between clients and servers. Also this protocol exists in a number of versions. We present in this report an analysis of several instances of the Transport Layer Security protocol.

1.3 Prerequisites

The reader of this thesis is supposed to have some knowledge of the concepts of cryptographic protocols. The reader should furthermore be familiar with the basic techniques of static analysis presented in [31]. Some understanding of semantics and its applications [33] is also required. The implementation of the techniques described in this report is carried out in Standard ML of New Jersey [40]. However this thesis does not require extensive knowledge in this area.

1.4 Structure of this Report

Chapter 2 introduces the concepts and usage of the SAML Single Sign-On protocol we wish to analyse in this thesis.

Chapter 3 introduces a formal way of describing protocols. Also we describe how implementations of protocols are deduced from specifications.

Chapter 4 the process calculi LySa used to describe the implementations of the protocols is introduced, as well as the static analysis used to validate security properties of the protocols. Last in the chapter we show how to use the analysis on a simple example.

Chapter 5 contains details on the extensions of the LySa-tool. This extension enables us to describe protocols using transport layer protocols on message transfers.

Chapter 6 presents the analysis of the SAML Single Sign-On protocol. The analysis results are obtained using the extended LySa-tool to analyse the described protocols.

Chapter 7 summarises our work, and concludes on the security aspects of the analysed protocols. Also possible improvements and alternative approaches are discussed in this chapter.

Chapter 2

Single Sign-On protocols

Today websites increasingly offer personalised content. This require users to authenticate themselves, which is usually done by giving the user an account with an associated user name and password. The website could be an online shop where users need to be authenticated to see their current orders. Other examples of websites could be: news sites, game sites or libraries. This means a user could, potentially, have loads of accounts, each on different websites. It is tedious for a user to remember many user names and passwords, and often a user will resort to using only a single user name and password for every website. From a security point of view, this is not a wise strategy. If a single login is eavesdropped and thereby compromised, all logins are compromised. Furthermore, it has a cost to the websites to manage the many accounts for their users. Each website need their own system to issue new accounts, remove unused accounts, handle lost passwords and other administrative tasks associated with the accounts. Also as accounts are often created ad-hoc, websites cannot truly rely on the identity information stated, when an account is created. Websites could be more strict and require valid identity information, but as it could exclude or scare some of their users this is often not an option.

These problems have motivated the development of new ways to authenticate users, of which Single Sign-On protocols are one. The purpose of these protocols are to allow users to authenticate themselves to several websites using *one* account, and if visiting more websites simultaneously or in quick succession, a single log on is sufficient. At the same time, when using a Single Sign-On protocol, each website does not have to manage accounts for users. Account management can be carried out at a central authentication site. As the users only have to use one account it should be easier to use and remember the associated password. Using a Single Sign-On protocol the user only should create one account, to authenticate themselves to the authentication site used by the websites providing the services.

This chapter describes how a Single Sign-On protocol functions, more specifically the SAML-Single Sign-On protocol. An example is used to demonstrate the scenario in with SAML SSO is used and how it works. The roles played in the protocol are discussed and finally two examples of real-life single sign-on protocols which extend SAML SSO are described.

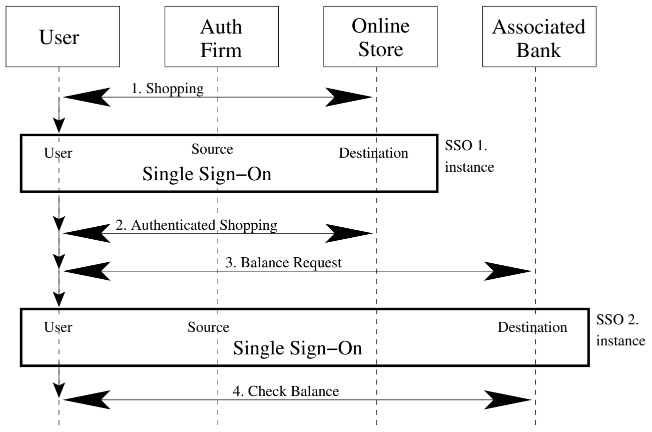


Figure 2.1: SSO example

2.1 SAML SSO

We have chosen to focus on the SAML Single Sign-On (SSO) protocol. This protocol is developed as a standard within the SAML (Security Assertion Markup Language) framework by the standardisation organisation OASIS [41]. The SAML-framework is used for secure exchange of information between online business partners.

The use of standard web-browsers as the client-program makes the SAML SSO protocol flexible and usable across different platforms, without the need to install additional software. To illustrate a typical scenario in which the SAML SSO protocol may be used let us consider Figure 2.1.

Assume a user visits an online store (message 1 of figure 2.1). At some point he must authenticate himself to the store and this may happen through a special authentication firm, by using the SSO protocol, as shown in the first instance of the protocol on the figure. The authentication firm[1] will ask the user to login using a password. As a result of the SSO protocol the authentication firm and the online store now know the identity of the user and the shopping can continue (message 2 on the figure). Later the user may like to look at the balance of his bank account (message 3). Since the bank does not know the user he must now authenticate himself to the bank. As before an SSO protocol is used for establishing the connection. The authentication firm recognises the user and authenticates him to the bank *without* requesting the password again. The user is now able to check the balance of his account (message 4). Notice that the user logs in only once at the authentication firm which enables him to gain access to various sites without needing to reauthenticate.

2.2 Roles & Concepts

In the scenario where the SAML Single Sign-On protocol is deployed principals are divided into three roles:

[1]The authentication firm should be trusted by all other principals prior to the run of the protocol, this is further explained in section 2.5 on page 13.

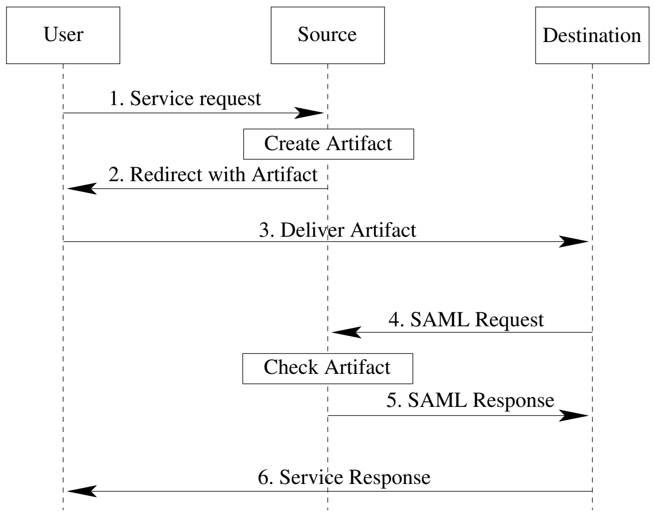


Figure 2.2: The SAML SSO

Users: The user accesses the network via a standard web-browser that implements standard protocols such as HTTP and TLS. The user is only linked to the web-browser while he is logged on a specific service.

Destinations: The destination site offers a restricted service which is only available to users authenticated by a central authentication server.

Sources: The source site authenticates users, thereby allowing users to access services at destination sites.

There are no restrictions on the number of principals of the three kinds; however each principal only act in one role.

The protocol uses the concept of an *artifact* that arises because most browsers only allow for a limited length of the HTTP URL. Many websites send arguments along with the HTTP query by appending them to the query. For example when a search for the string "test" is submitted to the search engine *google* the resulting URL is: `http://www.google.com/search?q=test`; in this example the URL is 36 characters long. If a more complicated argument was to be transferred this could become too long for the browser to handle. An artifact is a unique string which is used as an argument in an URL instead of the real data.

In the SAML scenario, the artifact is used as an identifier for a larger *SAML assertion*. An assertion is described in the SAML specification as a package of information that supplies one or more statements made by an authority [20, line 124–130]. These statements describe the identity of the subject, how and when the authentication was performed, details on how the subject could be re-authenticated, etc.

2.3 Protocol steps

Figure 2.2 describes the six steps of the SAML Single Sign-On protocol:

1. The user makes a HTTP request for a SAML assertion to a specific service at a restricted destination site.

2. If the source site recognises that the user is authenticated, an artifact and the corresponding SAML assertion are created. The source site responds to the users request by redirecting him to the desired destination site. The artifact is part of the redirect message enabling the user to identify himself to the destination site.

3. The user transparently completes the redirect started in step 2 by sending the artifact to the destination site.

4. In order to verify the identity of the user, the destination site sends a SAML:Request message stating the desired assertions and containing the artifact to the source site.

5. The source site receives the SAML:Request and looks up the SAML assertion using the artifact. The source site then sends a SAML:Response message back to the destination site with the answer to the requested assertions.

6. The SAML:Response contains information needed by the destination to be able to proceed the communication requested by the user. Depending on the contents of the SAML:Response message the destination site sends a HTTP response either allowing or denying access to the restricted service.

An artifact can only be used once and therefore the source site maintains a list of pending assertions. When an artifact is checked by the source site it and the associated assertions are removed from the pending list.

In a variant of the protocol the user enters the destination site prior to step 1. Then the user is redirected to the source site, the protocol continues as described above. We have chosen only to consider the version of the protocol where the source site is entered first, but it should be straight forward to apply our analysis to other versions of the protocol.

2.4 Possible Attacks on Protocols

The SAML specification states a list of *Security and Privacy Considerations* in [26]. In this document a number of possible attack scenarios on the SAML SSO protocol is described. At the same time possible counter-measures used as protection against attacks on the protocols are described. We will shortly describe each possible attack and how they could occur, as well as the recommendations from the specification [26].

The possibility of the attacks presented below have motivated the OASIS organisation [41] to add security premises to the SAML Single Sign-On protocol. We will later in this thesis verify whether they are sufficient or not.

Eavesdropping.

This attack occurs, when the attacker can intercept and read messages from the protocol, see Figure 2.3 on the next page. When the attacker acts as an eavesdropper, the attack is carried out in a *passive* way, meaning the attacker does not send any messages to principals in the protocol, and the principals of the protocol does not engage any communication with the attacker either. All other attack-types are *active* attacks.

Usually encryption is used to guarantee that eavesdropping cannot occur. It is important that the used encryption is guaranteed to be secure with respect to the attacker, since otherwise the attacker still is able to understand the intercepted message. The SAML specification [26] states that usage of TLS/SSL as transport layer should prevent eavesdropping.

Most of the complex attacks on known protocols include eavesdropping as a part of the attack, to gain basic knowledge.

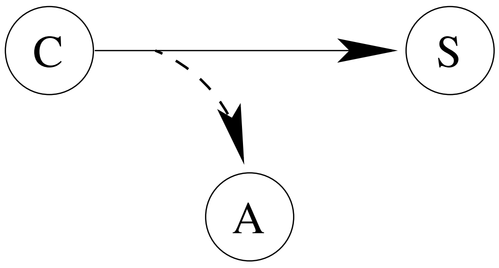


Figure 2.3: Eavesdropping

Replay attack

A replay attack occurs, when the attacker sends a message (or a part of a message) that has been sent in a previous run of the protocol to a principal in the current run of the protocol, see Figure 2.4. To avoid replay attacks protocols often try to verify a freshness property of messages. This could be done using nonces, timestamps and a counter initialised at a random number.

The SAML specification [26] again uses the TLS/SSL transport protocol as a countermeasure, since this protocol provide in-transit confidentiality allowing only the two principals who originally engaged communication to send messages on this transport layer, also a sequence number is added to each message.

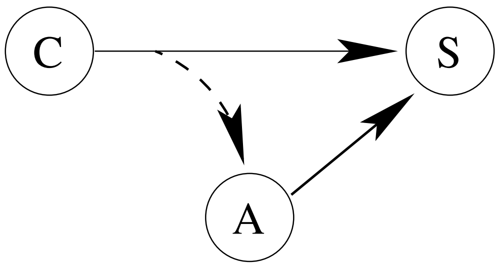


Figure 2.4: Replay attack

Modification

Modification attacks occur, when a message is intercepted by the attacker (eavesdropping), and then modified, see Figure 2.5. The attacker could modify the entire message, or just a singe field in the message. Encrypted data does not provide data integrity it self, since even though the message cannot be modified, it could be replaced with another message encrypted using the same key. To prevent this hashing is used. If a message and its corresponding hash value is signed using a private[2] key (asymmetric cryptography), the recipient of this message is able to verify the signature and the hash-value, doing so that data integrity is verified.

The SAML specification [26] uses the TLS protocol, since it guaranties in-transit data integrity.

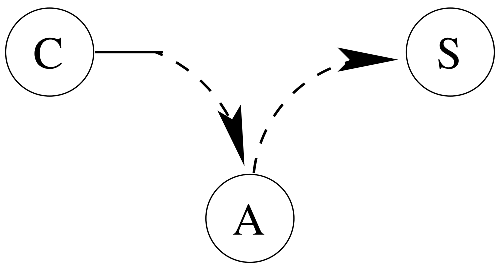


Figure 2.5: Modification

Man-In-the-Middle

A Man-in-the-Middle attack is a combination of an eavesdropping and a modification attack, where the attacker places himself between two communicating principals intercepting each of the communicated messages, possible altering them before sending them on to the intended principal, see Figure 2.6. In order to prevent such attacks a bilateral authentication is required, allowing the two communicating principals to verify that the received messages originates from the other principal.

The TLS protocol exists in a version, where bilateral authentication is used, therefore the SAML specification [26] uses TLS as transport layer.

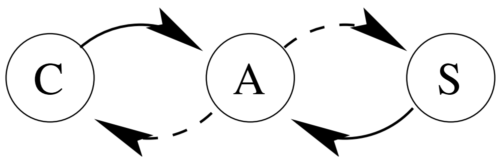


Figure 2.6: Man-In-the-Middle

[2]The private key must be a "private" key in the sense that a private key is only intended for one principal.

Deletion/Insertion attacks

These kinds of attacks occurs when an attacker deletes message from the protocol (deletion attack) without the principal noticing it. This will not result in a security breach, rather it would lead to a halt of the protocol. Since this is a implementation issue rather than a security issue this is not discussed.

The attacker could try to start the protocol by sending a request to a destination site, this would be an insertion attack. Since a simple request is no security threat to the protocol, this kind of attacks should not be considered. Insertion attacks are also covered by the combination of the already presented attacks.

Insider attack

A type of attacks not covered in the SAML security document [26] is insider attacks. An insider attack occurs when a previously trusted principal suddenly becomes a malicious attacker. In the worst case scenario this could lead to disclosure of all secrets this principal has shared with other principals in the scenario. Insider attacks for example occurs, when a security breach for one or more principals are not detected immoderately, and an attacker then is able to use this breach to attack the system from the inside. Usually such attacks are not taken in to consideration as they would often lead to a complete breakdown of security, though it could be insightful to examine the effect of a worst case security breach.

2.5 Public Key Infrastructure (PKI)

On the Internet users routinely have to verify the identity of a website and trust that it is the correct identity. This could be when a user is accessing an online shop or when using an online bank service. It is very important that a user can be sure he is not the victim of a Man-In-the-Middle attack. To cater for this a Public Key Infrastructure has been developed. The basic idea behind this is to have all web browsers trust a few companies acting as Certificate Authorities. Each CA has a root certificate meaning that web browsers know and trust this certificate. This puts a great deal of trust on the company acting as a CA and should a CA turn “bad” it would immediately loose this trust.A CA can issue certificates to websites and a user can verify these certificates by verifying that a trusted CA has issued it.

In a Single Sign-On scenario the principals involved must trust the Source-Site before the protocol can be used. The SAML specification [20] describes how to obtain this trust as:

> The primary mechanism is for the relying party and asserting party to have a pre-existing trust relationship, typically involving a Public Key Infrastructure.

The users and destination sites are relying principals trusting the asserting principal, in this case the source sites. The users are not required to have certificates as it would be an economic burden on the users. As a result the asserting principal must rely on other means of authenticating the users. The relying principal in fact rely on the asserting principal and, hence, does not have to trust the user directly.

The trust relationship between asserting and relying principals must be negotiated in advance. This is not a problem as it is done only once. Another issue is how to ensure that the principals can still trust each other. One principal might “turn bad” resulting in a broken protocol and this is not handled in the specification documents. The PKI infrastructure itself only deals with identity not trustworthiness.

2.6 Single Sign-On Applications

The problems addressed by a Single Sign-On protocol are common and the resolve to address them widespread. This has lead to different implementations of Single Sign-On protocols. Some are closed proprietary protocols and other open free standards. Some free SSO protocols are build on top of SAML SSO. Two major implementations on top of SAML SSO exist: the Liberty-Alliance Project [37] and the Shibboleth© Project [42]. These different implementations of the

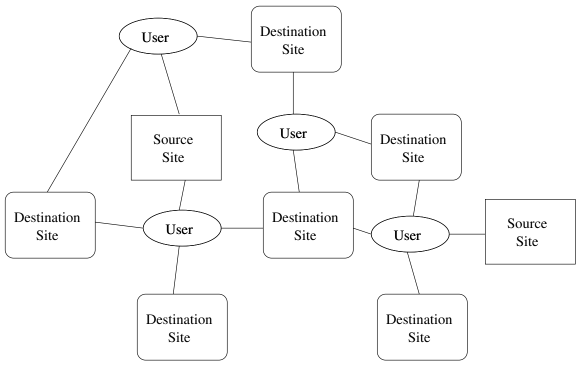


Figure 2.7: The SAML SSO scenario

Single Sign-On protocol, supply the protocol with additional features, to adjust the protocol to the specific scenario in which they are deployed. The features added stem from differing views on how the network and users are organised. To emphasise the differences we first take a look at the SAML scenario, shown in figure 2.7. In this scenario a number of different users exist, who all have access to at least one source site. Users can also access a number of different destination sites offering different services. The users need to be authenticated to the destination sites.

We shall give a short overlook of two different applications of the SAML SSO protocol, to show the different approaches which can be taken. The Single Sign-On protocol is only small parts of these projects, but since it is the main concern of this report we will only be interested in the implementation of this protocol. This will show the effort being put into the development of Single Sign-On protocols using SAML SSO and thereby further underlining the relevance of this report.

Liberty Alliance Project

[37] An alliance between more than 150 companies around the world supports the Liberty Alliance Project. The doctrine is to develop an open standard for federated network identity supporting current and emerging network devices. In the scenario envisioned by Liberty Alliance, a number of users exist, who all have access to a global network. On this global network a number of service providers and identity providers exist, corresponding to destination sites and source sites in the SAML scenario. The service providers and identity providers are organised in small "circles of trust". Inside these circles of trust, data provided by the user at one service provider can be shared among all the service providers using the identity provider as a hub for this communication. This could be the address information of a user, meaning that the user only has to type in his or her address once inside the current "circle of trust". This is what Liberty Alliance call federation of data, which is essentially sharing of data. The scenario is shown on figure 2.8. For this sharing of data to work it is of cause necessary for the user to trust all service and identity providers within the "circle of trust".

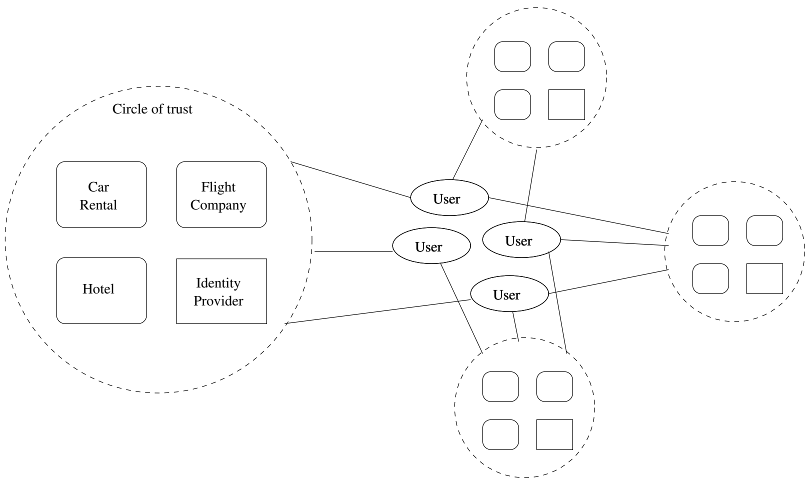


Figure 2.8: The Liberty Alliance scenario

To exemplify this assume that a user using the Liberty scenario is ordering a flight-ticket, and providing the flight company with the needed data such as name, address, phone number credit card number etc. When these data are submitted the user is asked to allow federation (sharing) of the data. Allowing this the user enters a site of a car rental company the user is transparently logged in via the Single Sign-On protocol. The user is now asked whether the car rental company should be allowed to view federated data. If the user allows this, the user is now able to rent a car and pay for it without providing additional data unless the data already provided is not sufficient. These properties applies to all service providers within the circle of trust the user has entered. The identity provider should store what service providers are allowed to read federated data so that the user should only allow this once.

Also the Liberty Alliance addresses the need to support many different mobile devices such as palm-pilots, mobile phones etc. This is however not completely new, since the SAML SSO is supported on any device able to run a modern web browser.

The management of federated data, such as allowing data to be federated, allowing service providers to read addresses, account numbers etc. is done by a number of different protocols developed by the Liberty Alliance organisation. To summarise the Liberty Alliance implementation of the Single Sign-On protocol enables service providers to gain access to federated data from the identity provider. When data are federated, and service providers are allowed to read the federated data, a simplified version of the Liberty Alliance SSO protocol would implement the same basic steps as SAML SSO presented in figure 2.2 on page 9. In stead of just a SAML:Response the message in step 5 also should contain the federated data concerning the user.

Shibboleth

[42] is developed and funded by a number of universities in cooperation with IBM. The goal of the Shibboleth project is to provide a link between existing campus authentication systems and resource providers offering services to universities. Shibboleth emphasise the privacy of the user. In some countries universities are legally required to protect the privacy of their students. Shibboleth does this by only allowing resource providers to see some general attributes of a user and not the identity of a user. The overall scenario is structured as shown in figure 2.9. The main idea is not very different from other Single

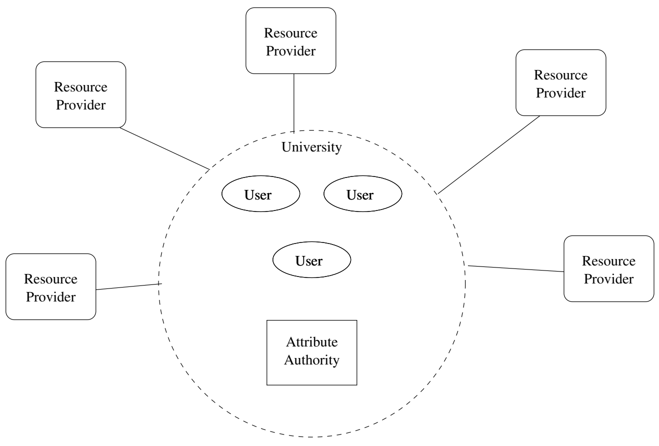


Figure 2.9: The Shibboleth scenario

Sign-On cases; a user enrolled on campus either as a student or as staff request access to a restricted resource, the resource site request some authentication, the authentication is done via an attribute authority present at campus.

As an example a biology student request access to a restricted resource, e.g. a database containing research material only accessible to biology students and professors. The student authenticates himself using the attribute authority present at the campus. The artifact received from this attribute authority is send on to the resource site. Now the resource site contacts the attribute authority at the campus, to verify the authenticity of the user, and if this is verified the user gain access to the restricted database. The student would be able transparently to use the Shibboleth Single Sign-On implementation to log on to another restricted resource, e.g. a journal collection requiring the user to be a masters student.

The Shibboleth implementation of the Single Sign-On protocol provides the same basic 6 steps as the SAML SSO, the main difference is however that users are structured in a hierarchy at the attribute authority, and the restricted resource is only verified that the user has certain attributes. In the previous example, the resource offering the restricted research database is only provided with the required attributes of the student, in this example that the student is a biology student. The steps 4 and 5 (SAML:Request and SAML:Response) in the SAML SSO is in the Shibboleth implementation substituted with what they call Attribute:Query and Attribute:Response.

Using federated administration, the Shibboleth implementation provides user privacy, in such a way that the resource site only know that it is a certain university that has accessed a resource. If a resource is misused the attribute authority is of cause able to link the used artifact back to a certain user. In this way the privacy of the user is protected alongside the need for efficient system administration.

Chapter 3

Specifying SAML SSO

The specification of the SAML SSO protocol does not contain any protocol narration in the classical sense. Some charts illustrating the message flow are presented but no abstraction from the complex message structures is given. This could be a result of the designers focussing more on the implementation of the protocols rather than a formal validation of the protocols. In the literature, security protocols are described using the standard notation of protocol narrations. We use an extended form of protocol narration named *extended protocol narrations* defined in *Automatic Validation of Protocol Narration* [8]. In this chapter we will develop an extended protocol narration for SAML SSO and for the underlying TLS transport protocol.

3.1 Protocol Narrations

In protocol narrations an exchange of a message sent from A to S, containing $A, B, \{K\}_{K_A}$ where $\{K\}_{K_A}$ denotes the name K encrypted under the key K_A is formalised as:

$$A \rightarrow S \ : \ A, B, \{K\}_{K_A}$$

This is the first message of the WMF protocol, which where also used in an example presented in Section 1.1 on page 1. We wish to use an extended version of this notation to formalise the protocols to be analysed. Extended protocol narrations differ from the classical protocol narrations in that they distinguish between the outputs and the corresponding inputs. This enables us to insert explicit checks on values and cryptographic operations, that would otherwise be left as implementation decisions. The source and destination of the messages are prepended to the message. This allows the receiving principal to check whether he was the intended receiver. Using this extension the first step of the WMF protocol would now be split into three pars:

$$\begin{array}{lll}
1. & A \rightarrow & : A, S, A, B, \{K\}_{K_A} \\
1.' & \rightarrow S & : x_A, x_S, x'_A, x_B, x_{Mess}, \quad [\texttt{check } x_S = S, x_A = x'_A] \\
1.'' & \rightarrow S & : \texttt{decrypt } x_{Mess} \texttt{ as } \{x_{Key}\}_{K_A}
\end{array}$$

First A sends a message on the network containing five parts: the identity of the sender (A), the intended recipient (S), the two principals to establish communication between (A, B)[1] and the session key encrypted with a shared longterm key $\{K\}_{K_A}$.

In step $1'$ we express that S receives the message and the variables x_A, x_S, x'_A, x_B and x_{Mess} are bound to the corresponding pars of the received message. To make sure that the message really is meant for S it is explicitly checked that x_S is bound to S ($x_S = S$), and it is checked that the principal initiating the communication is the one who wishes to establish secure communicate to B ($x_A = x'_A$).

In step $1.''$ S will decrypt $\{K\}_{K_A}$ bound to x_{Mess} and extract its contents using the variable x_{Key} which in turn will be bound to K.

We are interested in analysing authenticity properties of the protocol. This property depend on whether messages are sent and received by the correct principals in the protocol. To be able to capture this we annotate messages with a destination at the sender, and an origination at the receiver. Only encrypted messages are annotated as an attacker capable of eavesdropping on network traffic would be able to read any clear text messages. With this we get the following extended protocol narration for the first step of the WMF protocol:

$$\begin{array}{llll}
1. & A \rightarrow & : A, S, A, B, \{K\}_{K_A}\,[\texttt{dest}\;\{S\}] & \\
1.' & \rightarrow S & : x_A, x_S, x'_A, x_B, x_{Mess} & [\texttt{check}\;x_S = S, x_A = x'_A] \\
1.'' & S & : \texttt{decrypt}\;x_{Mess}\;\texttt{as}\;\{x_{Key}\}_{K_A}\,[\texttt{orig}\;\{A\}] &
\end{array}$$

The $[\texttt{dest}\;\{S\}]$ in line 1 means that the encrypted message $\{K\}_{K_A}$ should only be decrypted at the principal S, and the $[\texttt{orig}\;\{A\}]$ denotes that the message decrypted in line $1.''$ should have been encrypted at principal A.

WMF Using this formalism on the all the message in the WMF protocol as presented in Section 1.1 on page 1 we get:

$$\begin{array}{llll}
1. & A \rightarrow & : A, S, A, B, \{K\}_{K_A}\,[\texttt{dest}\;\{S\}] & \\
1.' & \rightarrow S & : x_A, x_S, x'_A, x_B, x_{Mess} & [\texttt{check}\;x_S = S, x_A = x'_A] \\
1.'' & S & : \texttt{decrypt}\;x_{Mess}\;\texttt{as}\;\{x_{Key}\}_{K_A}\,[\texttt{orig}\;\{A\}] & \\
2. & S \rightarrow & : S, x_B, x_A, \{x_{Key}\}_{K_B}\,[\texttt{dest}\;\{B\}] & \\
2.' & \rightarrow B & : y_S, y_B, y_A, y_{Mess} & [\texttt{check}\;y_B = B] \\
2.'' & B & : \texttt{decrypt}\;y_{Mess}\;\texttt{as}\;\{y_{Key}\}_{K_B}\,[\texttt{orig}\;\{S\}] & \\
3. & A \rightarrow & : A, B, \{m_1, ..., m_k\}_K & \\
3.' & \rightarrow B & : y'_A, y'_B, y_{Messages} & [\texttt{check}\;y'_B = B, y'_A = y_A] \\
3.'' & B & : \texttt{decrypt}\;y_{Messages}\;\texttt{as}\;\{y_{m_1}, ..., y_{m_k}\}_{y_{Key}} &
\end{array}$$

Table 3.1: Extended protocol narration of WMF

[1]This part of the message could be simplified to contain only the identity of B since the first part of the message shows that the identity of the initiator is A

```
<element name="Request" type="samlp:RequestType"/>
<complexType name="RequestType">
  <complexContent>
    <extension base="samlp:RequestAbstractType">
      <choice>
        <element ref="samlp:Query"/>
        <element ref="samlp:SubjectQuery"/>
        <element ref="samlp:AuthenticationQuery"/>
        <element ref="samlp:AttributeQuery"/>
        <element ref="samlp:AuthorizationDecisionQuery"/>
        <element ref="saml:AssertionIDReference" maxOccurs="unbounded"/>
        <element ref="samlp:AssertionArtifact" maxOccurs="unbounded"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
```

Table 3.2: SAML:Request

```
<element name="Response" type="samlp:ResponseType"/>
<complexType name="ResponsetType">
  <complexContent>
    <extension base="samlp:ResponseAbstractType">
      <sequence>
        <element ref="samlp:Status"/>
         <element ref="saml:Assertion" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Table 3.3: SAML:Response

```
<element name="SubjectConfirmation" type="saml:SubjectConfirmationType"/>
<complexType name="SubjectConfirmationType">
  <sequence>
    <element ref="saml:ConfirmationMethod" maxOccurs="unbounded"/>
    <element ref="saml:SubjectConfirmationData" minOccurs="0"/>
    <element ref="saml:ds:KeyInfo" minOccurs="0"/>
  </sequence>
</complexType>
```

Table 3.4: SubjectConfirmation

We have already explained the message: 1., 1.′ and 1.″. The two other messages follow in a similar fashion. Notice that the variables introduced in 1′ and 1.″ are used by S again in 2. when it sends a message to B. Similarly variables known used by B in 2.″ and 2.″ are used again in 3.′ and 3.″. We will use this notation style to describe SAML SSO in a formal fashion.

3.2 Message Format

In section 2.3 on page 9 the flow of the SAML Single Sign-On protocol is described as 6 basic steps. To extend these steps into an extended protocol narration the format of the messages should be clear. The format of the messages determines the implementation, therefore the considerations made must be well thought through. In this section we shall give a reasoning about the format of the messages in our model of the SAML SSO protocol.

Since SAML is a framework, and not an implementation of the Single Sign-On protocol, we have to consider many implementation issues. The SAML specification [24, 25] specifies a number of XML-schemes used to validate the contents of SAML-messages. We shall start by describing of how, in particular the SAML:Request and SAML:Response messages are used and ow we have modelled them.

The protocol uses a very general form of assertions that can record many different aspects about the user. As an example, the destination site can use a SAML:Request in step 4 in the protocol to ask the source site how the user was authenticated as well as requesting a key for secure communication with the user. The source site then responds with the information requested in SAML:Request using a SAML:Response message.

The XML-scheme for a SAML:Request message is shown in table 3.2. The <*choice*> element indicates that a valid SAML message only have to include one of the elements listed in its body. The elements themselves each refer to other XML documents which again might contain <*choice*> elements and refer to yet other documents in this way creating a large set of possible messages. The choice of elements depend on what information (assertion) the destination site requires. In our formalisation of the protocol we shall ignore this detailed structure and only model the artifact embedded in the SAML:Request message and the user's key in the SAML:Response. This corresponds to assuming that the source and destination sites beforehand have agreed always to use *one* specific form of request.

In table 3.3 the overall XML-scheme for SAML:Response message is shown and it also rely on other XML documents describing the details. This is the format of the message used to respond to the SAML:Request messages and like the request, the response also has a large set of possible messages. If the destination site receives a SAML:Response from the source site we shall in our formalisation assume that the validation of the artifact was successful. To model the response, the source site sends the identity of the user and the cryptographic key held by the user to the destination site. A cryptographic key is encoded in the SAML:Response using the *SubjectConfirmation* XML-scheme shown in table 3.4 on the preceding page. This scheme contains a *saml:ds:KeyInfo* element which hold the cryptographic key. The *SubjectConfirmation* scheme is not directly part of the SAML:Response scheme; actually it is a subelement to one of the subelements of SAML:Response.

Also the encoding of the artifact should be considered. It is encoded as a sequence of bytes. A part of this sequence is an *AssertionHandle* implemented as a random 20 byte sequence. This *AssertionHandle* must according to the specification [25] be *infeasible to construct or guess.* If it is impossible to guess the AssertionHandle it should also be impossible to guess the Artifact, therefore we shall in our model treat the artifact as a nonce.

SAML SSO

Using this message format, and applying the methods about extended protocol narration from section 3.1 on page 19 we produce the extended protocol narration for the SAML Single Sign-On protocol in table 3.5 on the facing page. Here we write D for the destination site, S for the source site and U for the user. It should be straightforward to identify the six steps of the protocol from figure 2.2. Since no encryption is added to the model of the SAML SSO protocol at this stage, the first 5 original steps are each only extended to a sending part $x.$ and a receiving part $x.'$. The last step contains encrypted information, and therefore it is extended into three parts, the last part being the decryption $6.''$. To ensure confidentiality and authenticity of the messages some security mechanisms must be added to the message transfers.

$$\begin{array}{lll}
1. & U \rightarrow & : U, S, D \\
1.' & \rightarrow S & : x_U, x_S, x_D \quad [\texttt{check}\ x_S = S] \\
2. & S \rightarrow & : S, x_U, x_D, Artifact \\
2.' & \rightarrow U & : y_S, y_U, y_D, y_{Artifact} \quad [\texttt{check}\ y_U = U] \\
 & & \quad [\texttt{check}\ y_S = S,\ y_D = D] \\
3. & U \rightarrow & : U, y_D, S, y_{Artifact} \\
3.' & \rightarrow D & : z_U, z_D, z_S, z_{Artifact} \quad [\texttt{check}\ z_D = D] \\
4. & D \rightarrow & : D, z_S, z_{Artifact} \\
4.' & \rightarrow S & : x_D, x_S, x_{Artifact} \quad [\texttt{check}\ x_S = S] \\
 & & \quad [\texttt{check}\ x_{Artifact} = Artifact] \\
5. & S \rightarrow & : S, D, K_U, U \\
5.' & \rightarrow D & : z'_S, z_D, z_{Key}, z_{U'} \quad [\texttt{check}\ z_D = D] \\
 & & \quad [\texttt{check}\ z_{U'} = z_U] \quad [\texttt{check}\ z'_S = z_S] \\
6. & D \rightarrow & : D, U, \{Mess\}_{z_{Key}}\, [\texttt{dest}\ \{U\}] \\
6.' & \rightarrow U & : y_D, y_U, yc_{Mess} \quad [\texttt{check}\ y_U = U] \\
6.'' & U & : \texttt{decrypt}\ yc_{Mess}\ \texttt{as}\ \{y_{Mess}\}_{K_U}\, [\texttt{orig}\ \{D\}]
\end{array}$$

Table 3.5: Extended protocol narration for SAML SSO

3.3 Protocol Layers

The communication in the SAML SSO protocol takes place on a public network possibly the Internet, using the standard communication protocols for this kind of communication. On figure 3.1 the protocol stack used in SAML SSO is shown. In a protocol stack the data has to travel from the bottom to the top, to be

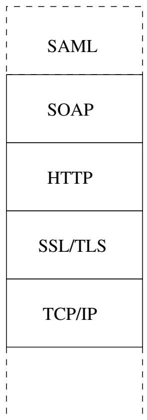


Figure 3.1: The protocol stack used in SAML

received, and the other way to be send. The layers beneath TCP/IP layer, are the layers providing the physical link to the network. These layers does not provide any security features to the actual protocol and is therefore not modelled. The following sections describe very briefly how the layers work and how we represent them in our model of the SAML SSO protocol.

3.3.1 TCP/IP

The TCP/IP layer is the network/transport layer. This layer does the actual delivery of messages. A simple abstraction of this layer is found in the extended protocol narration in that the first element of a message is the sender and the second element is the intended receiver. For example given the following protocol narration:

$$Source \rightarrow Destination \quad : Payloaad$$

using extended protocol narration this would be transformed to:

$$Source \rightarrow \qquad : Source, Destination, Payloaad$$

Since the *Source* and *Destination* are sent in clear-text, the message could easily be intercepted by an attacker, and modified.

3.3.2 TLS

Transport Layer Security This layer can provide authentication and confidentiality on messages. All modern browsers implements the TLS protocol, as well as all web servers. This makes the protocol one of the most commonly used protocols to transfer data secure data over the Internet since browsers are the client program for web servers.

In SAML SSO TLS is used to provide security properties to the messages in the protocol. The TLS protocol is rather complicated, and therefore we return to the problem doing a qualified abstraction in section 3.4 on page 26.

3.3.3 HTTP

The *HyperText Transport Protocol* is used to transport World Wide Web pages between a web server and a browser. The SAML SSO protocol is using the HTTP protocol to transfer data between the principals. We do not model HTTP directly but rather we model the content of the messages. In this context HTTP is used mainly to instruct the browser of what to do next. HTTP has a way to send errors back to browsers but as we do not model errors we will not describe them here. Each paragraph in the following describes a method from the HTTP-protocol [15] used in the SAML SSO protocol:

GET The GET method is used to query a web server for information. This could be a request for a home page on the Internet. The request can include parameters in the HTTP-URL. For example when a search for the string "test" is submittet to the search engine *google* the resulting URL is: `http://www.google.com/search?q=test`. The characters to the right of the "?" is parameters for the query. In this example the URL is 36 characters long but could be much longer. Most browsers has a limit of how long an URL can be.

This is one of the reasons for using artifacts in SAML SSO protocol as mentioned in Section 2.2 on page 8. We model the data in the query resulting in the following protocol narration:

$$\begin{array}{llll}
1. & C \rightarrow & : C, S, Query & \\
1.' & \rightarrow S & : x_C, x_S, x_{Query} & [\texttt{check } x_S = S] \\
2. & S \rightarrow & : S, C, Response & \\
2.' & \rightarrow C & : y_S, y_C, y_{Response} & [\texttt{check } y_C = C]
\end{array}$$

In step 1., the Client C sends a *Query* to the Server S. In step 1.′ the *Request* is received and processed by the Server. If the server recognises the *Request* it creates a corresponding *Response*. In step 2. the *Response* is sent by the Server, and it is received in step 2.′ by the user.

REDIRECT This method is used by the server to send the browser to a new location. The REDIRECT message contains the URL of the new location. A REDIRECT method could very well be a valid response to a GET query if the Server is requested for a service not available, but the server has the knowledge of another server offering this service. When a browser receives a REDIRECT message it performs a GET query with the URL received in the REDIRECT message. Notice this URL contains both the new server and the request string. This is modelled as:

$$\begin{array}{llll}
1. & S \rightarrow & : S, C, S', Request' & \\
1.' & \rightarrow C & : y_S, y_C, y_{S'}, y_{Request'} & [\texttt{check } y_C = C] \\
2. & C \rightarrow & : C, y_{S'}, y'_{Request} & \\
2.' & \rightarrow S' & : x_C, x'_S, x_{Request} & [\texttt{check } x'_S = S'] \\
3. & S' \rightarrow & : S', C, Response & \\
3.' & \rightarrow C & : y'_S, y_C, y_{Response} & [\texttt{check } y_C = C]
\end{array}$$

In step 1. the server sends the name of the new server S' as well as the new *Request′* to the Client. This is received by the Client in step 1.′. Now the Client uses the GET method to query the new location $y_{S'}$ with the new request $y_{Request}$. The steps 2 and 3 are really just a GET with the server $y_{S'}$ and the request $y'_{Request}$.

POST The POST method has the same effect as the GET method, but the data send to the server is send in a message body in stead of encoded into the HTTP-URL, which allows the POST method to transfer unlimited length of data to the Server. The POST method is usually used in cases where large amounts of data must be updated. The difference between POST and GET methods are that a REDIRECT will instruct a browser to issue a new GET query. This mean POST cannot be used in conjunction with REDIRECT. Besides this there is no difference in how POST and GET is modelled.

3.3.4 SOAP

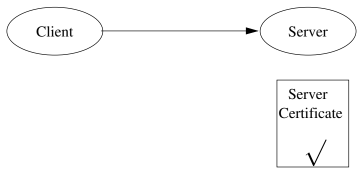
Simple Object Access Protocol [29] is used to exchange data in a decentralised distributed scenario. It is fundamentally just a structured way to exchange messages using XML. From our point of view it is only used to define the message formats. In section 3.2 on page 21 we have discussed how the abstraction from the specified XML-structure to protocol narration i done, therefore there should be no need of doing further modelling of the specific message format.

3.4 Transport Layered Security

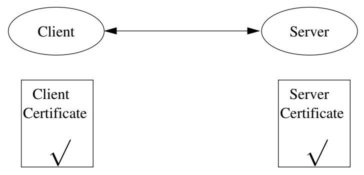
'The documents describing the SAML SSO protocol recommend using the Transport Layer Security (TLS) protocol for establishing secure communication [13]. TLS is based upon the SSL 3.0 protocol [4] and is supported by almost all Internet browsers for ensuring secure communication between users and websites. We shall return to the actual recommendations of the specification [25] in Section 6.2 on page 83 and focus on the TLS protocol itself in the present section.

3.4.1 TLS

The TLS protocol is used in a scenario where a client wishes to establish secure communication with a server. TLS can be used in two ways (shown on Figure 3.2); either in an unilateral version where the server has a certificate issued and signed by a mutually trusted Certificate Authority (CA); or in the bilateral version the client has a similar certificate. In the unilateral version the Server is authenticated to client but not the other way around as it is the case in the bilateral version. The protocol use certificates to prove the identity of the prin-



(a) Unilateral TLS



(b) Bilateral TLS

Figure 3.2: The TLS variants

cipals. Then a common master key is agreed upon. All further communication is exchanged using session keys derived from this master key.

The TLS protocol has two layers, the *record* protocol and on top of that, a layer using one of four protocols [13]. The record protocol handles messages sent on the network, and as part of this, it can fragment and compress the messages before they are sent. Furthermore, messages may have a Message Authentication Code (MAC, see 3.4.2 on page 30) applied and they may be encrypted. At the receiving part, data is then decrypted, verified using the MAC code, decompressed and reassembled. The four protocols used on top of the TLS record layer are: a *handshake* protocol, an *alert* protocol, a *change cipher spec* protocol and an application data protocol:

- **The handshake protocol** is by far the most complex of the four protocols. It negotiates the encryption keys and verifies the certificates.
- **The alert protocol** consists of a single message used to signal failure of the protocol.
- **The change cipher spec protocol** consists of a single message which instructs the receiving principal that subsequent messages will be using the newly negotiated cipher spec.
- **The application data protocol** is essentially a transparent protocol. When this protocol is used it is only the record layer which is in effect.

The TLS protocol is summarised in Table 3.6 using a notation close to the one found in [13]. The left column lists the messages sent by the client while the right column show those sent by the server; the order of the exchanges is from top to bottom, so ClientHello will be the first message exchanged.

The table focuses on the four protocols mentioned above and thus ignores the details of the record layer. However, several messages will be grouped together and could be handled as a single message by the record protocol; as an example the five messages ServerHello, Certificate, ServerKeyExchange, CertificateRequest and ServerHelloDone are treated as a single message by the record protocol. The

Client		**Server**
ClientHello	$\longrightarrow$	
		ServerHello
		ServerCertificate
		ServerKeyExchange*
		CertificateRequest*
	$\longleftarrow$	ServerHelloDone
ClientCertificate*		
ClientKeyExchange		
CertificateVerify*		
[ChangeCipherSpec]		
Finished	$\longrightarrow$	
		[ChangeCipherSpec]
	$\longleftarrow$	Finished
Application Data	$\longleftrightarrow$	Application Data

Table 3.6: Message flow for a full handshake

messages marked with an asterisk (*) are optional as they will only be needed in specific runs of the protocol. In the following we shall discuss the individual messages in more detail:

ClientHello This is the first message sent in the handshake protocol. It can be sent at any time by the client requesting a new session or resuming an old session. The message contains a time stamp, 28 bytes of random data, a list of supported compression methods and a list of cipher methods the client is willing to use. In case the client wishes to resume a session, a *SessionID* is included in the message. The use of session resume can shorten the handshake protocol but the Finished messages (see below) will always be sent. [13, chap. 7.4.1.2]

ServerHello This message contains a time stamp, 28 bytes of random data, a session id, the selected cipher method and the selected compression method. [13, chap. 7.4.1.3]

ServerCertificate If the selected cipher method is not anonymous, the server sends it's certificate. The certificate can be one of several different types but it must be trusted by the client. [13, chap. 7.4.2]

ServerKeyExchange It may be the case, that the connection is not anonymous and the client cannot use the certificate to encrypt the premaster secret and to cater for this situation the server sends a public key that can be used instead. [13, chap. 7.4.3]

CertificateRequest The server may request a certificate to authenticate the client by sending this message. It contains a list of acceptable certificate types and certificate authorities. [13, chap. 7.4.4]

ServerHelloDone This message signals that the server is done sending the initial response messages. When the client receives this message it should start processing the messages from the server. [13, chap. 7.4.5]

ClientCertificate This message contains the client's certificate but it is only sent by the client if the server has requested a certificate. [13, chap. 7.4.6]

ClientKeyExchange The client generates a *premaster secret* which consists of 46 bytes of random data. It is crucial to keep this value secret. It is sent to the server encrypted with the servers certificate or the public key provided in ServerKeyExchange of the previous messages. Alternatively the premaster secret could be generated by using the Diffie Hellman Key Agreement protocol. In this case the present message is left empty. [13, chap. 7.4.7]

CertificateVerify This message is only sent if a ClientCertificate message was sent by the server and it should enable the server to verify that the received certificate belongs to the client. The message is constructed by taking a hash of all previous messages exchanged in the handshake protocol and signing it with the clients certificate. [13, chap. 7.4.8]

[ChangeCipherSpec] This message is sent using the *Change cipher spec* protocol and it informs the other principal that the next messages will be sent using the newly negotiated cipher specs. A similar message also have to be sent by the server before exchanging the Finished messages. The effect is that the sender changes his sending part of the record protocol, and the receiver changes the receiving part of the record protocol to use the new encryption and mac.

Finished(client) This message signals the end of the above sequence of messages sent by the client and it is used to verify that both client and server have agreed on the same master secret. The message contains a hash of all exchanged messages. [13, chap. 7.4.9]

Finished(server) This message is then used to confirm that both client and server have agreed on the same master secret — however this is not exactly the same as Finished(client) as it will take the additional message into account. [13, chap. 7.4.9]

Before sending the *ChangeCipherSpec* message the session and MAC keys are calculated. TLS uses a custom hash function called *Pseudo Random Function* (PRF). It is hash function combining the output from *SHA1*[1] and *MD5*[35]. When running the *handshake* protocol both principals will learn the value of the time stamps and random values in the hello messages and the premaster secret. These values are then combined and hashed using *PRF* to obtain the master secret. All session and MAC keys are generated from this. The keys generated are: read_MAC, write_MAC, read_key and write_key. Additionally initialisation vectors for the ciphers are generated.

3.4.2 Considerations Modelling TLS

Clearly, the above description contains many details that cannot be modelled. We shall now discuss how to obtain a model of TLS. It is important to remember the assumption made in Section 1.2 on page 4, that the encryption used in the model is perfect, meaning that the only reverse function of encryption is decryption with the correct key. Also it is assumed that encrypted values hold enough redundant data to provide data integrity. We have no notion of time in our model and consequently time stamps cannot be modelled. Also error handling hardly makes sense as either the protocol is stuck or it finishes all steps successfully.

Hash functions and key generation. We can model a hash function using a public name as the key of an asymmetric encryption. Since there does not exist a corresponding key to the public name it is neither a public nor a private key (a more thorough explanation will be given in section 4.5 on page 49 . This has the effect that the encrypted value can never be decrypted as no decryption key exist. Still it allows for matching of the value. The hash function used in TLS is called PRF (Pseudo Random Function) and is modelled by using PRF as a key in an asymmetric encryption. This is exemplified by the following extended protocol narration:

$$\begin{array}{lll}
1.\ A \rightarrow & : A, B, Mess, \{|Mess|\}_{PRF} & \\
1.' \quad \rightarrow B & : x_A, x_B, x_{Mess}, x_{Hash} & [\texttt{check}\ x_A = B, \{|x_{Mess}|\}_{PRF} = x_{Hash}]
\end{array}$$

A sends in step 1. a message $Mess$ and the hash value calculated from the message $\{|Mess|\}_{PRF}$ to principal B. To be able to verify the hash value in step 1.$'$ principal B must calculate the value from the received message x_{Mess} and compare it to the received hash-value x_{Hash}, this is done as follows: $\{|x_{Mess}|\}_{PRF} = x_{Hash}$.

Message Authentication Codes. In TLS a keyed MAC (HMAC, see [21]) is used to verify the integrity of messages. This is modelled using a shared secret key and a cryptographic hash function. The message is hashed along with the key and a sequence number. This hash value can then be used to verify the integrity of the message.

MAC codes which are used for the sole purpose of data integrity can be omitted, since we have made the assumption that encrypted values hold enough redundant data to provide data integrity. As we shall see later the sequence numbers cannot be left out as they ensure messages within a single session cannot be confused with one another. This is modelled by using a sequence of public values *Seq1*, *Seq2*, ... and each message will be encrypted along with one of these numbers using the current session key. The i'th message transfer from principal A to principal B will be:

1. $A \rightarrow \ : A, B, \{Seq_i, Mess\}_{K_{Session}}$
1.′ $\rightarrow B : x_A, x_B, x_{EncryptedMess}$ [`check` $x_B = B$]
1.″ $B :$ `decrypt` $x_{EncryptedMess}$ `as` $\{x_{Seq}, x_{Mess}\}_{K_{Session}}$ [`check` $x_{Seq} = Seq_i$]

To be able to do the check $x_{Seq} = Seq_i$ the server and the client using the TLS connection to communicate must agree upon when to start the sequence numbering.

Renegotiation and cipher specs. When using TLS both the server and the client can request renegotiation of keys and ciphers at any time. This is implemented to allow the change to a stronger cipher before sending more sensitive information or to renew old keys.

This is not modelled, the reason being that we model perfect cryptography and new keys are negotiated using the old secure connection.

Session resume. A client can resume a session by sending a *ClientHello* with the old *SessionID*. If both principals still believe the master secret is secret, then new random values are exchanged using the old session keys. For simplicity we do not model this.

Signatures Signatures are modelled using the private key of a principal. If a message is encrypted using a private key then the message is signed, since the all principals can get hold of the public key and thereby decrypt it. But only the owner of the private key is able to encrypt a message using this key. The use of signatures is illustrated in the following example:

1. $A \rightarrow \ : A, B, \{|Mess|\}_{K_A^-}$
1.′ $\rightarrow B : x_A, x_B, x_{SignedMess}$ [`check` $x_B = B$]
1.″ $B :$ `decrypt` $x_{SignedMess}$ `as` $\{|x_{Mess|}\}_{K_A^+}$

Principal A sends a message signed using the private key K_A^- to principal B, to be able to read the signed message principal B decrypts the message using the corresponding public key K_A^+. To be able to read the signed message principal

B must have the knowledge of the public key. If a principal is able to read a signed message using a public key, the message is verified to be signed by the holder of the corresponding private key.

In cryptographic protocols signatures are often used alongside with symmetric encryption, in these cases the symmetric encryption is used to preserve confidentiality and the asymmetric encryption (signatures) are used to authenticate the principals in the protocol. It is important to note the difference between a message where a *Secret* first is signed using the private key K^- and thereafter encrypted using a secret key K_{Secret}:

$$\{\{|Secret|\}_{K^-}\}_{K_{Secret}}$$

and a message where the *Secret* first is encrypted using the secret key and thereafter signed using the private key:

$$\{|\{Secret\}_{K_{Secret}}|\}_{K^-}$$

Even seen from a perspective where both the public key, K^+ and the secret key, K_{Secret} is known these two messages have entirely different meanings. The first message indicates that the sender has knowledge of the secret key, and the original message, and that the signature is correct. The second message only assures that the sender knows the private key K^-. The original message could be intercepted from another principal and the original signature removed, and a new one added without decrypting the original message. This feature is emphasised as one of the principles in [5]:

> *Sign before encryption. If a signature is affixed to encrypted data, then one cannot assume that the signer has any knowledge of the data. A third party certainly cannot assume that the signature is authentic, so non repudiation is lost.*

To be able to use signatures, there must exist a trust relationship between the two principals communicating; the public key must have been shared in some way, possibly by an existing public key infrastructure (PKI) using a known key distribution protocol, or by using certificates.

Certificates and Diffie Hellman. TLS can be set up to run without authentication of the server using anonymous key exchange([13, chap. F.1.1.1.]). This does not provide any protection against man-in-the-middle attacks and because the model is used only where authentication is needed we shall not model this. Rather we model TLS using unilateral authentication or bilateral authentication.

In the TLS scenario two principals exists, the server, S, and the client, C. To model certificates all principals will know CA^+, which is the public key of the certificate authority. In order to create certificates all honest clients also know CA^-; this is essentially the same as having all certificates issued by a single CA with the only exception being that also a dishonest client should be able to get a certificate. This can be achieved by having an honest client issue a certificate to the attacker. The attacker must not have the knowledge of the private key of the CA (CA^-), since then it would be able to falsify certificates and use these to engage attacks on the protocols. A certificate is modelled using the private public key pair of the principal. The certificate contains the identifier of the principal and the public key of the owner, and this is all signed by the CA. The following message shows the format of a certificate belonging to the server S:

$$\{|S, K_S^+|\}_{CA^-}$$

Certificates are used, when there is no existing trust relationship between the principals that are going to communicate. Instead of sharing public keys, they share the trust in the certificate authority. The public key of a principal is part of the certificate. To illustrate the use of certificates consider this example:

1.	$S \rightarrow \quad : S, C, \{|Mess|\}_{K_S^-}, \{|S, K_S^+|\}_{CA^-}$	
1.′	$\rightarrow C : x_S, x_C, x_{SignedMess}, x_{Certificate}$	[`check` $x_C = C$]
1.″	C : `decrypt` $x_{Certificate}$ `as` $\{|x'_S, x_{KS^+}|\}_{CA^+}$	[`check` $x'_S = x_S$]
1.‴	C : `decrypt` $x_{SignedMess}$ `as` $\{|x_{Mess}|\}_{x_{KS^+}}$	

In step 1. a server S sends a signed message to the client C, since there is no existing trust relationship between the two principals the server appends its certificate to the message. In step 1.′ the client receives the message and bind the values to the variables: x_S, x_C, $x_{SignedMess}$ and $x_{Certificate}$. If the client trusts the certificate authority he can decrypt the certificate in step 1.″. When the certificate belonging to the server is decrypted, the identifier of the server S is bound to the variable x'_S and the public key of the server is bound to the variable x_{KS^+}. After decryption of the certificate in step 1.‴, the client is able to read the signed message using the servers public key bound to x_{KS^+}. At this stage it is verified to the client, that the message $Mess$ is in fact signed by the holder of the certificate.

3.4.3 Extended Protocol Narration

We shall now present extended protocol narrations for the unilateral and bilateral versions of TLS; they are given in Tables 3.7 on the next page and 3.8 on page 34 and explained below.

Unilateral TLS

The extended protocol narration for TLS with unilateral authentication is shown in Table 3.7 on the next page. The first message is ClientHello which is modelled by a nonce N_c; recall that time stamps, compression, different cipher methods and session resumes are not modelled so the only part left is the random value. It is received by S in 1.′ and here the server check the message is intended for it. Because the server does not know which client it is communicating with it only stores its name in x_c.

Message 2 contains the messages ServerHello, Certificate and the empty ServerHelloDone. As above the ServerHello message is modelled using a nonce. The certificate $\{|S, KS^+|\}_{CA^-}$ belonging to the server is sent in the Certificate message and it is received by the client which can check that the message is intended for it (using $y_c = C$), and that is comes from S (using $y_s = S$). Also the client checks the certificate by decrypting it with CA^+ and ensuring that $y_s = S$. At the same time the client obtains the server's public key y_{KS}.

1.	$C \rightarrow$		$: C, S, N_c$
1.′		$\rightarrow S$	$: x_c, x_s, x_{Nc}$ [`check` $x_s = S$]
2.	$S \rightarrow$		$: S, x_c, N_s, \{\lvert S, KS^+\rvert\}_{CA^-}$
2.′		$\rightarrow C$	$: y_s, y_c, y_{Ns}, y_{cert}$ [`check` $y_s = S, y_c = C$]
2.″		C	: `decrypt` y_{cert} `as` $\{\lvert y_s, y_{KS}\rvert\}_{CA^+}$ [`check` $y_s = S$]
3.	$C \rightarrow$		$: C, S, \{\lvert pm\rvert\}_{y_{KS}}, \{Seq1, \{\lvert\{\lvert N_c, y_{Ns}, pm\rvert\}_{PRF}\rvert\}_{PRF}\}_{sessionkey}$ [`dest` $\{S\}$]
3.′		$\rightarrow S$	$: x_c, x_s, x_{cpm}, x_{mh}$ [`check` $x_s = S$]
3.″		S	: `decrypt` x_{cpm} `as` $\{x_{pm}\}_{KS^-}$
3.‴		S	: `decrypt` x_{mh} `as` $\{x_{Seq1}, x_{rmh}\}_{sessionkey}$
			[`check` $x_{Seq} = Seq1, x_{rmh} = \{\lvert\{\lvert x_{Nc}, N_s, x_{pm}\rvert\}_{PRF}\rvert\}_{PRF}$]
4.	$S \rightarrow$		$: S, x_c, \{Seq2, \{\lvert\{\lvert x_{Nc}, N_s, x_{pm}\rvert\}_{PRF}\rvert\}_{PRF}\}_{sessionkey}$
4.′		$\rightarrow C$	$: y_s, y_c, y_{mhash}$ [`check` $y_s = S, y_c = C$]
4″		C	: `decrypt` y_{mhash} `as` $\{y_{Seq2}, y_{rmhash}\}_{sessionkey}$ [`orig` $\{S\}$]
			[`check` $y_{Seq2} = Seq2, y_{rmhash} = \{\lvert\{\lvert N_c, y_{Ns}, pm\rvert\}_{PRF}\rvert\}_{PRF}$]

Table 3.7: TLS unilateral authentication

Before message 3 is sent the client generates a new premaster secret pm. From the premaster secret, N_c and y_{Ns} the master secret is generated as $\{|N_c, y_{Ns}, pm|\}_{PRF}$. The ClientKeyExchange message contains the premaster secret encrypted with the server's public key. After this the ChangeCipherSpec message should be sent; however, we shall merely assume that the server know when to change to the new cipher and omit this message in the model. Finally, the Finished message is sent over the new cipher. The resulting Finished message is encrypted together with a sequence number $Seq1$ using the session key generated from the master secret. The Finished message should be a hash of all previous messages but we limit it to be a hash of the master secret. The messages are received by S and the encrypted premaster secret is decrypted to x_{pm}. This allows the server to calculate the session key and verify the Finished message x_{mh}.

The fourth and last message of the protocol is the server's Finished message to the client. It is modelled as a hash of the master key $\{|\{|x_{Nc}, N_s, x_{pm}|\}_{PRF}|\}_{PRF}$ and is sent using the encrypted connection just established. It is encrypted with the session key along with the second sequence number $Seq2$ and will be verified by the client. Notice here the ChangeCipherSpec message is also omitted.

Bilateral TLS

To model bilateral authentication a few changes are needed. Table 3.8 on the following page show the extended protocol narration for this instance of TLS. Compared with the unilateral version of Table 3.7 only message changed is message 3. It now contains the clients certificate ($\{|C, KC^+|\}_{CA^-}$) together with the CertificateVerify message. The latter is constructed along the same lines as the Finished messages, by hashing the master secret and signing it with the clients certificate ($\{|\textit{master secret}|\}_{KC^-}$). These extra messages are verified by the server leading to two more decryptions at $3''$ and $3''''$.

The protocol narrations of Tables 3.7 and 3.8 only show the handshake proto-

1.	$C \rightarrow$	$: C, S, N_c$
$1.'$	$\rightarrow S$	$: x_c, x_s, x_{Nc}$ $[\texttt{check}\ x_s = S]$
2.	$S \rightarrow$	$: S, x_c, N_s, \{\|S, KS^+\|\}_{CA^-}$
$2.'$	$\rightarrow C$	$: y_s, y_c, y_{Ns}, y_{cert}$ $[\texttt{check}\ y_s = S, y_c = C]$
$2.''$	C	$: \texttt{decrypt}\ y_{cert}\ \texttt{as}\ \{\|y_s, y_{KS}\|\}_{CA^+}$ $[\texttt{check}\ y_s = S]$
3.	$C \rightarrow$	$: C, S, \{\|C, KC^+\|\}_{CA^-}, \{\|pm\|\}_{y_{KS}}, \{\|\{\|\{\|N_c, y_{Ns}, pm\|\}_{PRF}\|\}_{PRF}\|\}_{KC^-},$
		$\{Seq1, \{\|\{\|N_c, y_{Ns}, pm\|\}_{PRF}\|\}_{PRF}\}_{sessionkey}$ $[\texttt{dest}\ \{S\}]$
$3.'$	$\rightarrow S$	$: x_c, x_s, x_{cert}, x_{cpm}, x_{certv}, x_{mh}$ $[\texttt{check}\ x_s = S]$
$3.''$	S	$: \texttt{decrypt}\ x_{ccert}\ \texttt{as}\ \{c_{cc}, x_{KC}\}_{CA^+}$ $[\texttt{check}\ x_c = x_{cc}]$
$3.'''$	S	$: \texttt{decrypt}\ x_{cpm}\ \texttt{as}\ \{x_{pm}\}_{KS^-}$
$3.''''$	S	$: \texttt{decrypt}\ x_{certv}\ \texttt{as}\ \{\|x_m\|\}_{x_{KC}}$
		$[\texttt{check}\ x_m = \{\|\{\|x_{Nc}, N_s, x_{pm}\|\}_{PRF}\|\}_{PRF}]$
$3.'''''$	S	$: \texttt{decrypt}\ x_{mh}\ \texttt{as}\ \{x_{Seq1}, x_{rmh}\}_{sessionkey}$ $[\texttt{orig}\ \{x_c\}]$
		$[\texttt{check}\ x_{Seq} = Seq1, x_{rmh} = \{\|\{\|x_{Nc}, N_s, x_{pm}\|\}_{PRF}\|\}_{PRF}]$
4.	$S \rightarrow$	$: S, x_c, \{Seq2, \{\|\{\|x_{Nc}, N_s, x_{pm}\|\}_{PRF}\|\}_{PRF}\}_{sessionkey}$ $[\texttt{dest}\ \{x_c\}]$
$4.'$	$\rightarrow C$	$: y_s, y_c, y_{mhash}$ $[\texttt{check}\ y_s = S, y_c = C]$
$4.''$	C	$: \texttt{decrypt}\ y_{mhash}\ \texttt{as}\ \{y_{Seq2}, y_{rmhash}\}_{sessionkey}$ $[\texttt{orig}\ \{S\}]$
		$[\texttt{check}\ y_{Seq2} = Seq2, y_{rmhash} = \{\|\{\|N_c, y_{Ns}, pm\|\}_{PRF}\|\}_{PRF}]$

Table 3.8: TLS bilateral authentication

col but more messages can be sent from either side. Such messages will all take the general form $\{Seq_i, Message\}_{sessionkey}$, with a corresponding decryption operation and with Seq_i being a unique sequence number for the i'th message in the communication.

Crypto-points

The narrations have been annotated with crypto-points as a preparation for doing an analysis of the security properties. The crypto-points are only of use after the handshake protocol has established the session key. As the session key is secret encryptions or decryptions using it can be annotated with crypto-points. In the case of unilateral authentication crypto-points are only meaningful on the client side because the client is not authenticated to the server. On the other hand in the case of bilateral authentication all encryptions and decryptions can be annotated. In the unilateral case this can be seen in Table 3.7 on the page before message 3. and $4.''$. Any messages sent using the already established TLS connection should also be annotated on the client side. In the bilateral case Table 3.8 show this in the messages: 3., $3.'''''$, 4. and $4.''$. Any further message should be annotated on both the client and the server side.

3.5 Assembling the Model

We have now derived a model for the SAML SSO protocol, and for two versions of the TLS protocols used to apply security to the message transfers. In the final description of the entire SAML SSO model the extended protocol narrations for the two TLS versions have to be inserted in the SAML SSO protocol narration.

The extended protocol narrations for the TLS protocols should be inserted parameterised in the SAML SSO protocol, since the SAML SSO protocol uses several different instances of the TLS protocol. For each instance of a TLS protocol some *initialisation* is done, thereafter a parameterised version of the extended protocol narrations from either Table 3.7 or 3.8 should be used to establish communication. When the message transfers using a TLS connection are done, the messages should be encrypted using the session key belonging to the current TLS connection, and a sequence number should be added.

The insertion of a parameterised version of the TLS-protocol into the SAML SSO protocol is a very tedious and error prone work, if it should be done by hand therefore this shall wait as we later in Chapter 5 on page 63 describe an automated method for inserting at transport layer protocol into another protocol.

Chapter 4

Process calculus

To conduct a formal analysis of the SAML SSO protocol we need to have a formalised description of the protocol. One common way to do formal descriptions is to use a process calculus. A substantial number of different process calculi has been developed over the years.

Communicating sequential processes in short CSP [19] was one of the early process calculi. In CSP one can define parallel processes synchronising/communicating using events. Later the π calculus [27, 28] introduced the notion of channels. This meant that instead of using events to communicate, communication was done over channels. Channel could be secret between some processes meaning that only they could send and receive messages on them. The SPI or Secure π calculus introduces encryptions and decryptions. Now a key can be used to encrypt a message and the only by knowing the key it would be possible to read the message. This closely resembles the way protocols using encryption work.

LySa [8] is using some of the elements from SPI/π to form a process calculus with encryption and decryption but with only one global channel or network. Messages sent from one process to another are conceptually communicated using a network everyone can see. The syntax of LySa match the syntax of extended protocol narrations closely. The translation between the two is very straight forward.

4.1 LySa

In order to analyse the SAML SSO protocol we shall first formalise it in the process calculus LySa [8]. Here we present a short list of the characteristics of LySa we will be using.

- There is only one *global ether* for communication and hence local communication is excluded.
- The calculus has primitives for *symmetric and asymmetric cryptography*; the encryption operations are formalised as part of the term language of the calculus.
- Decryption is modelled using *pattern matching* which also plays a central

role for communication where it can be used to filter messages sent on the ether.

In LySa terms are used to describe processes. Terms are built from names (including symmetric keys, nonces, etc.), variables and public and private keys. They can be composed using symmetric and asymmetric encryption. The syntax of terms E is shown in table 4.1. Here $\mathcal{N}$ and $\mathcal{X}$ denote sets of names and

$E ::=$	*terms*	
	n	name ($n \in \mathcal{N}$)
	x	variable ($x \in \mathcal{X}$)
	k^+, k^-	public and private keys
	$\{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}]$	symmetric encryption ($k \geq 0$)
	$\{\!\vert E_1, \cdots, E_k \vert\!\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}]$	asymmetric encryption ($k \geq 0$)

Table 4.1: LySa terms

variables. Encryptions are tuples of terms $\{E_1, \ldots, E_k\}$ encrypted under the term E_0. The term E_0 can be constructed from other terms it this way forming a complex key. We assume that encryption is perfect, meaning that the only inverse function of encryption is to use decryptions with the correct key. The

$P ::=$	*processes*	
	0	nil
	$\langle E_1, \cdots, E_k \rangle.\, P$	output
	$(E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k).\, P$	input (with matching)
	$P_1 \mid P_2$	parallel composition
	$(\nu\, n)\, P$	restriction
	$(\nu_{\pm}\, m)\, P$	key pair creation
	$!\, P$	replication
	$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k\}^{\ell}_{E_0}\ [\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$	symmetric decryption (with matching)
	$\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\vert E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k \vert\!\}^{\ell}_{E_0}\ [\mathsf{orig}\,\mathcal{L}]\ \mathsf{in}\ P$	asymmetric decryption ($k \geq 0$)

Table 4.2: The LySa process language

process language contains the usual operations for the inactive process, parallel composition, restriction and replication; however, note that restriction can also be used to introduce a key pair for asymmetric cryptography. The output and input operations are polyadic; however, the input operation is combined with pattern matching meaning that it will only succeed if a prefix of the message matches the terms specified in the input operation. The receiving process $(E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k).\, P$ means that a tuple containing the first j received values is pairwise matched to the tuple of j terms $(E_1, \cdots, E_j)$, if there is a pairwise coresponence the remaining $k - j$ values of the received message are bound to the variables $x_{j+1}, \cdots, x_k$. This is syntacticaly indicated using a semicolon to seperate the terms to be matched and the variables to be bound. This pattern matching is also used in decryptions. The syntax of processes is shown i Table 4.2.

As an example consider the first communication of the WMF protocol:

$$A \to S : \quad A, B, \{K\}_{K_A}$$

This communication was divided into three parts in the extended protocol narration in section 3.1 on page 19:

$$\begin{array}{lll} 1. & A \to \quad & : A, S, A, B, \{K\}_{K_A} \\ 1.' & \quad \to S & : x_A, x_S, x'_A, x_B, x_{Mess}, \qquad [\texttt{check}\ x_S = S, x_A = x'_A] \\ 1.'' & \quad \to S & : \texttt{decrypt}\ x_{Mess}\ \texttt{as}\ \{x_{Key}\}_{K_A} \end{array}$$

The coresponding LySa process shlould contain the same 3 parts, principal A sends the message (1.), ther server S receives the process (1.') and the server S decrypts the message (1."). The process for the principal A will contain the send action

$$\langle A, S, A, B, \{K\}_{K_A} \rangle. \cdots$$

whereas the process for S will contain the receive action

$$(A, S, A; x_B, x_{Mess}). \cdots$$

thereby enforcing not only that a tuple containing five parts is received but also that the first three components are A, S and A, respectively. There are no requirements on components four and five of the message, they will be bound to the variables x_B and x_{Mess}. To decrypt messages, LySa contains explicit decryption operations that are also combined with pattern matching. Continuing the above example, the process for S would be continued with:

$$(A, S, A, x_B; x_{Mess}).\ decrypt\ x_{Mess}\ as \{; x_{Key}\}_{K_A}\ in\ \cdots$$

thereby requiring that A and S agree on the key K_A. The decrypted value of the encrypted message x_{Mess} in S is bound to the variable x_{Key}. In this example there is no matching going on in the decrypt operations, since there are no terms to the left of the semi-colon.

In order to express the intentions of the protocols the LySa syntax includes annotations about the origin and destination of messages. Encryptions can be annotated with a crypto-point defining it's position in the process. To state where the encrypted value is intended to be decrypted a set of destination crypto-points can be added. Similarly, each decryption construct can be annotated with a crypto-point defining its position in the process as well as a set of crypto-points specifying the potential origins of the encrypted messages. For the first communication of the WMF protocol, the corresponding LySa code looks as follows:

$$\begin{array}{l} (\nu\, K_A)\,(\\ (\nu\, K) \\ \langle A, S, A, B, \{K\}_{K_{AB}} [\mathsf{at}\, A\ \mathsf{dest}\, S\,] \rangle.\mathsf{0} \\ | \\ (A, S, A;\ xB, xMess). \\ \mathsf{decrypt}\ xMess\ \mathsf{as}\ \{;\ xKey\}_{K_A} [\mathsf{at}\, S\ \mathsf{orig}\, A\,]\ \mathsf{in}\, \mathsf{0} \\ ) \end{array}$$

The construct $[\mathsf{at}\, A\ \mathsf{dest}\, S\,]$ in process A specifies that the encryption is created at crypto-point A and is intended for decryption at crypto-point S. The process for S specifies that the message to be decrypted at crypto-point S is intended to come from crypto-point A, using the corresponding statement $[\mathsf{at}\, S\ \mathsf{orig}\, A\,]$.

The restriction constructs are used to limit the scope of the keys so that an attacker does not know them. A further explanation on why restrictions are necessary is presented in section 4.4 on page 47. The two restrictions in this example have different scope. The restriction of the existing longterm key K_A between principal A and the Server $(\nu\, K_A)$ has a scope that includes both the Server and principal A. This is indicated using parentheses around rest of the LySa process. The session key K is created by principal A, and therefore a scope only including principal A. This is seen as no parentheses follow the restriction $(\nu\, K)$.

In table 4.3 the entire LySa process for the WMF protocol is listed. The number in the left margin refer to the message number of the extended protocol narration for WMF (see Table 3.1 on page 20). Step 0. in the LySa process is used to setup the longterm keys between the server S and the principals A and B. The scope of the restriction of K_A and K_B include the definition of both A and B and the Server S. Both the steps 1. and 3. of the principal A contains two lines. In the first case it is because the newly created key has to be restricted, and in step 3. the message containing confidential data must also be restricted. The matching and variable bindings in the LySa process is not done entirely as described in the extended protocol narration, we will explain this later in section 4.8 on page 58.

0.	$(\nu\, K_A)\,(\nu\, K_B)\,($
1.	$(\nu\, K)$
	$\langle A, S, A, B, \{K\}_{K_A}[\mathsf{at}\, A_1\ \mathsf{dest}\, S1\,]\rangle.$
3.	$(\nu\, Secret)$
	$\langle A, B, \{Secret\}_K\rangle[\mathsf{at}\, A_2\ \mathsf{dest}\, B\,]$
	$\vert$
1.′	$(A, S, A;\ xB, xMess).$
1.″	$\mathsf{decrypt}\ xMess\ \mathsf{as}\ \{;\ xKey\}_{K_A}[\mathsf{at}\, S1\ \mathsf{orig}\, A_1\,]\ \mathsf{in}$
2.	$\langle S, xB, A, \{xKey\}_{K_B}\rangle[\mathsf{at}\, S2\ \mathsf{dest}\, B1\,].0$
	$\vert$
2.′	$(S, B;\ yA, yMess).$
2.″	$\mathsf{decrypt}\ yMess\ \mathsf{as}\ \{;\ yKey\}_{K_B}[\mathsf{at}\, B1\ \mathsf{orig}\, S1\,]\ \mathsf{in}$
3.′	$(yA, B;\ yMessages).$
3.″	$\mathsf{decrypt}\ yMessages\ \mathsf{as}\ \{;\ ySecret\}_{yKey}[\mathsf{at}\, B2\ \mathsf{orig}\, A_2\,]\ \mathsf{in}\ 0$
)	

Table 4.3: LySa process for WMF

4.2 Semantics of LySa

To be precise on what a LySa process describes we shall in this section give a short description of the reduction semantics defined for LySa[8]. We use the notation of $P[E/x]$ to describe that all occurrences of x in process P should be replaced by the term E, (or simply the value of E is bound to variable x in P). The notation $\llbracket E_i \rrbracket$ is the term E_i with all destination/origin annotations removed. Names used in a LySa process are global in the sense that if a name "X" occurs in two places in the process they have the same meaning. This also apply to variables and it is therefore not possible to use local variables. In essence this means that each name should only be used in one meaning. For example should variable names only be used once. To simplify definition of the control flow analysis in section 4.3 on page 43, all occurrences of a bound name n is mapped to *one* canonical name $\lfloor n \rfloor$. The same mapping applies for variables. The function applied to terms $\lfloor E \rfloor$ replaces all names and variables in the term with their canonical versions. To ease the reading of the semantic rules we write E for the canonical term *canonE*. We say that two processes are α-equivalent only if the mapping of names and variables correspond.

Before we look at the semantics we present the structural congruence, $\equiv$, which LySa processes satisfy:

- $P \equiv Q$ if P and Q are disciplined α-equivalent;
- $P \mid Q \equiv Q \mid P$
  $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
  $P \mid \mathsf{0} \equiv P$
- $(\nu\, n)\, \mathsf{0} \equiv \mathsf{0}$,
  $(\nu\, n)\, (\nu\, n')\, P \equiv (\nu\, n')\, (\nu\, n)\, P$, and
  $(\nu\, n)\, (P \mid Q) \equiv P \mid (\nu\, n)\, Q \quad \text{if } n \notin \mathsf{fn}(P)$;
- $!\, P \equiv P \mid !\, P$

The rules for the reduction semantics $\rightarrow_{\mathcal{R}}$ is shown in Table 4.4 on the following page. The congruence should be used to reorder a process so it is possible to apply one of the rule from the reduction semantics.

Communication

The communication rule from Table 4.4 expresses that an output $\langle E_1, \cdots, E_k \rangle.\, P$ is matched by a corresponding input $(E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k).\, Q$ if the first j elements match pairwise; e.g. E_i with all annotations removed is compared to E'_i with all annotations removed for all i: $\wedge_{i=1}^{j} \llbracket E_i \rrbracket = \llbracket E'_i \rrbracket$. If the matching is successful rest of the terms $E_{j+1}, \cdots, E_k$ are bound to the variables $x_{j+1}, \cdots, x_k$.

Decryption

The rule for decryption expresses the matching on the term

$$\mathsf{decrypt}\ \{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\, \mathcal{L}]\ \mathsf{as}\ \{E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k\}^{\ell'}_{E'_0}\ [\mathsf{orig}\, \mathcal{L}']\ \mathsf{in}\ P$$

can only take place if the key used to decryption corresponds to the one used to create the encrypted term, this is expressed by $\llbracket E_0 \rrbracket = \llbracket E'_0 \rrbracket$ for symmetric decryption. For asymmetric decryption the key used for decryption must be the opposite of the one used to encrypt, expressed by $\{E_0, E'_0\} = \{m^{\pm}, m^{\mp}\}$, which is a short notation for $\{E_0, E'_0\} = \{m^{+}, m^{-}\} \vee \{E_0, E'_0\} = \{m^{-}, m^{+}\}$.

(Communication)

$$\frac{\wedge_{i=1}^{j} \llbracket E_i \rrbracket = \llbracket E'_i \rrbracket}{\langle E_1, \cdots, E_k \rangle .\, P \;\mid\; (E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k).\, Q \rightarrow_{\mathcal{R}} P \;\mid\; Q[E_{j+1}/x_{j+1}, \cdots, E_k/x_k]}$$

(Decryption)

$$\frac{\wedge_{i=0}^{j} \llbracket E_i \rrbracket = \llbracket E'_i \rrbracket \quad \wedge \quad \mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})}{\begin{array}{c}\mathsf{decrypt}\ \{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]\ \mathsf{as}\ \{E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k\}^{\ell'}_{E'_0}\ [\mathsf{orig}\,\mathcal{L}'\,]\ \mathsf{in}\ P \\ \rightarrow_{\mathcal{R}} P[E_{j+1}/x_{j+1}, \cdots, E_k/x_k]\end{array}}$$

(Asymmetric-Decryption)

$$\frac{\wedge_{i=1}^{j} \llbracket E_i \rrbracket = \llbracket E'_i \rrbracket \quad \wedge \quad \{E_0, E'_0\} = \{m^{\pm}, m^{\mp}\} \quad \wedge \quad \mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})}{\begin{array}{l}\mathsf{decrypt}\ \{\!|E_1, \cdots, E_k|\!\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]\ \mathsf{as}\ \{\!|E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k|\!\}^{\ell'}_{E'_0}\ [\mathsf{orig}\,\mathcal{L}'\,]\ \mathsf{in}\ P \\ \quad \rightarrow_{\mathcal{R}} P[E_{j+1}/x_{j+1}, \cdots, E_k/x_k]\end{array}}$$

(Parallel)

$$\frac{P \rightarrow_{\mathcal{R}} P'}{P \mid Q \rightarrow_{\mathcal{R}} P' \mid Q}$$

(Restriction)

$$\frac{P \rightarrow_{\mathcal{R}} P'}{(\nu\, n)\, P \rightarrow_{\mathcal{R}} (\nu\, n)\, P'}$$

(Asymmetric-Restriction)

$$\frac{P \rightarrow_{\mathcal{R}} P'}{(\nu_{\pm}\, m)P \rightarrow_{\mathcal{R}} (\nu_{\pm}\, m)P'}$$

(Congruence)

$$\frac{P \equiv Q \;\wedge\; Q \rightarrow_{\mathcal{R}} Q' \;\wedge\; Q' \equiv P'}{P \rightarrow_{\mathcal{R}} P'}$$

Table 4.4: Operational semantics, $P \rightarrow_{\mathcal{R}} P'$, parameterised on $\mathcal{R}$.

These strong requirements to the keys model the assumption about perfect encryption; data can only be decrypted using the correct key. The matching in the decryption construction take place as in the communication, where the first j terms are matched $\wedge_{i=1}^{j} \llbracket E_i \rrbracket = \llbracket E'_i \rrbracket$ and the last $k - j$ terms are bound to variables $E_{j+1}/x_{j+1}, \cdots, E_k/x_k$. The condition $\mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})$ is universally true and can therefore be ignored at the moment. We shall return to this condition on crypto-points in section 4.2 on the next page.

Parallel

The reduction rule for parallel construction is straight forward; two parallel processes $P \mid Q$ is reduced using the reduction semantics on either one of them e.g. $P \rightarrow_{\mathcal{R}} P$.

Restriction

As for the parallel construction, the reduction rule for the restriction construction $(\nu\, n)\, P$ applies the reduction semantics on the restricted process e.g. $P \rightarrow_{\mathcal{R}} P$. The same rule applies for restriction on asymmetric keys.

Congruence

The congruence rule in the reduction semantics, is not a rule about congruence, but the rule expresses that two congruent processes $P \equiv Q$ are reduced to two congruent processes $P' \equiv Q'$ if the reduction semantics are applied to them.

Reference Monitor Semantics

We have now defined a reduction semantics $\rightarrow_{\mathcal{R}}$, that produces relations for the annotations on encryptions and decryptions $\mathcal{R}(\ell, \mathcal{L}', \ell', \mathcal{L})$ but does not use them. We now look at another variant of the semantics, where the annotations are used to define a reference monitor semantics $\rightarrow_{\mathsf{RM}}$. The reduction rules of the semantics are the same, but instead of defining the $\mathcal{R}$ relation, the reference monitor takes this relation as input: $\mathsf{RM}(\ell, \mathcal{L}', \ell', \mathcal{L}) = (\ell \in \mathcal{L}' \wedge \ell' \in \mathcal{L}$. This means that the encryption should have made at l must be in the set $\mathcal{L}'$ of expected origins of the data, as well as the actual place where the decryption takes place l' must be in the set of expected destinations $\mathcal{L}$.

The idea of a *reference monitor* is to ensure that access to a specific object or data is only done in a valid way. In this case the reference monitor ensures that decryptions can only occur at places described in the annotations of the encryption. Otherwise the execution should be halted by the reference monitor. This reference monitor is used in the definition of the analysis of LySa-processes as the result contain a set describing where the reference monitor would have halted the process.

4.3 Static analysis

We now present a formal method to do a security analysis of LySa processes. The idea of the analysis is to track messages communicated on the network, along with the possible values of the variables in the protocol. Additionally the analysis will record the potential violations of the destination/origin annotations. In Figure 4.1 it is shown how the analysis approximate traces of a LySa

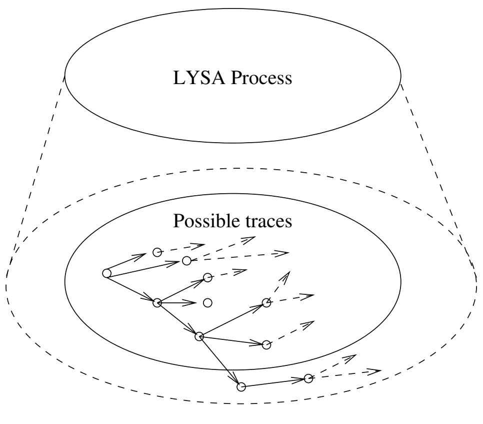


Figure 4.1: The over-approximation of a LySa-process

process. For example a LySa process describes a set of possible operations, to do a safe approximation of this, the analysis uses an over-approximation of this set, therefore the analysis could investigate a trace not possible at all.

4.3.1 Terms

In Table 4.5 the analysis for terms is given. The analysis of terms uses the notion of an abstract environment for names and variables:

$$\rho : \lfloor \mathcal{X} \rfloor \to \wp(\mathcal{V})$$

The abstract environment ρ maps the canonical variables $\lfloor \mathcal{X} \rfloor$ to the set of canonical values $\mathcal{V}$. The analysis of terms uses the abstract environment to make a judgement of the form:

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

Here $\vartheta \subseteq \mathcal{V}$ is a safe approximation of the set of values that E may evaluate to in the environment ρ. The rules in Table 4.5 expresses that ϑ must contains all the canonical values associated the components of a term. The rules for names and private/public key pairs say that the canonical names must be in ϑ. The rule for variables expresses that the set of canonical value the canonical variable maps to from the environment must be a subset of ϑ: $\rho(\lfloor x \rfloor) \subseteq \vartheta$. The encryption term $\{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]$ is constructed using a number of other terms $E_0, \cdots, E_k$. The rules of the analysis expresses that each term is analysed $\wedge_{i=0}^{k} \rho \models E_i : \vartheta_i$ and all combinations of values from this analysis $\forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \vartheta_i$ must be in ϑ belonging to the analysis of the overall encryption term $\{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta$. The notation $V \in \vartheta$ tests if V is in the set ϑ.

$$\frac{\lfloor n \rfloor \in \vartheta}{\rho \models n : \vartheta} \qquad \frac{\rho(\lfloor x \rfloor) \subseteq \vartheta}{\rho \models x : \vartheta} \qquad \frac{\lfloor m^+ \rfloor \in \vartheta}{\rho \models m^+ : \vartheta} \qquad \frac{\lfloor m^- \rfloor \in \vartheta}{\rho \models m^- : \vartheta}$$

$$\frac{\begin{array}{c}\wedge_{i=0}^{k} \rho \models E_i : \vartheta_i \ \wedge \\ \forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \vartheta_i \ \Rightarrow \ \{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta\end{array}}{\rho \models \{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,] : \vartheta}$$

$$\frac{\begin{array}{c}\wedge_{i=0}^{k} \rho \models E_i : \vartheta_i \ \wedge \\ \forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \vartheta_i \ \Rightarrow \ \{\!|V_1, \cdots, V_k|\!\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta\end{array}}{\rho \models \{\!|E_1, \cdots, E_k|\!\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,] : \vartheta}$$

Table 4.5: Analysis of terms, $\rho \models E : \vartheta$.

4.3.2 Processes

In the analysis of processes focus on what values can flow on the network, this is stored in an environment:

$$\kappa \subseteq \wp(\mathcal{V}^*)$$

This environment includes all messages communicated on the global network. Using the two environments ρ and κ the rules for processes are defined in Table 4.6 on the following page. The rules for processes have the form:

$$(\rho, \kappa) \models_{\mathsf{RM}} P : \psi$$

The analysis uses the reference monitor defined in section 4.2 on page 43, in such a way that if the reference monitor abborts, the annotation leading to the abbortion should be placed in the *error component* ψ and the execution should continue. The analysis returns a tuple of (ρ, κ, ψ) so the rules of the analysis are satisfied. The error component ψ contains an over-approximation of the potential origin/destination violations. If $(l, l') \in \psi$ it means that something encrypted at l could unexpectedly be decrypted at l'.

The analysis of processes use the analysis of terms defined in Table 4.5 on the facing page, where ϑ gives the set of values the terms can evaluate to. The rules for inactive processes $\mathsf{0}$, restriction $(\nu\, n)\, P$, parallel construct $P_1|P_2$ and replication $!\,P$ are all trivial, and does only require that the rules are satisfied for another simple process, therefore these rules are not explained further.

Output

The rule for sending a message on the network $\langle E_1, \cdots, E_k\rangle. P$ requires that each of the k terms are evaluated to a set of possible values using the rules for terms: $\wedge_{i=1}^{k}\ \rho \models E_i : \vartheta_i$, and all k-tuples of values $(V_1, \cdots, V_k)$ taken from $\vartheta_1 \times \cdots \times \vartheta_k$ can occur on the network κ, expressed as: $\forall V_1, \cdots, V_k :\ \wedge_{i=1}^{k} V_i \in \vartheta_i \ \Rightarrow\ \langle V_1, \cdots, V_k\rangle \in \kappa$. If the rules are satisfied for the output construct the rules should also be applied to the rest of the process $(\rho, \kappa) \models_{\mathsf{RM}} P : \psi$.

Input

The rule for input evaluates the first j terms $E_1, \cdots, E_j$ as to their acceptable estimates ϑ_i using the rules for terms, and checks whether the first j values of any message on the network $\langle V_1, \cdots, V_j, V_{j+1}, \ldots, V_k\rangle \in \kappa$ are pointwise included[1] in ϑ_i. If this is the case, the rest of the values from the message $V_{j+1}, \cdots, V_k$ are added to the acceptable estimate for the variables $x_{j+1}, \cdots, x_k$. If the communication succeeds, the continues process P is analysed.

Decryption

In the rule for decryption all terms are evaluated to their respectable estimates ϑ_i. The first j values $V_0, \cdots, V_j$ of the evaluation of the encrypted term $\{V_1, \cdots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\,\mathcal{L}\,] \in \vartheta$ are checked whether they are pointwise included in ϑ_i. For symmetric decryption the check $V_0 \mathrel{\mathsf{E}} \vartheta_0$ ensures that only the correct key can be used to decrypt encrypted values. For asymmetric decryption the rule expresses that the key used for decryption must be the opposite as the one used for encryption $\{V_0, V_0'\} = \{m^{\pm}, m^{\mp}\}$. If the matching succeeds for the first j values and the keys for decryption matches the ones used for decryption, the values $V_{j+1}, \cdots, V_k$ are added to the acceptable estimates for the variables $x_{j+1}, \cdots, x_k$. Finally the reference monitor is considered; if there is a violation

[1]The check $V_i \mathrel{\mathsf{E}} \vartheta_i$, describes that annotations are ignored as for the semantics rules

$$(\rho,\kappa) \models_{\mathsf{RM}} \mathsf{0} : \psi \qquad \frac{(\rho,\kappa) \models_{\mathsf{RM}} P : \psi}{(\rho,\kappa) \models_{\mathsf{RM}} (\nu\, n)\, P : \psi} \qquad \frac{(\rho,\kappa) \models_{\mathsf{RM}} P : \psi}{(\rho,\kappa) \models_{\mathsf{RM}} (\nu_{\pm}\, m) P : \psi}$$

$$\frac{(\rho,\kappa) \models_{\mathsf{RM}} P_1 : \psi \ \wedge\ (\rho,\kappa) \models_{\mathsf{RM}} P_2 : \psi}{(\rho,\kappa) \models_{\mathsf{RM}} P_1 | P_2 : \psi} \qquad \frac{(\rho,\kappa) \models_{\mathsf{RM}} P : \psi}{(\rho,\kappa) \models_{\mathsf{RM}} !\, P : \psi}$$

(Output)

$$\frac{\begin{array}{c} \wedge_{i=1}^{k}\ \rho \models E_i : \vartheta_i\ \wedge \\ \forall V_1, \cdots, V_k :\ \wedge_{i=1}^{k}\ V_i \in \vartheta_i\ \Rightarrow\ \langle V_1, \cdots, V_k \rangle \in \kappa\ \wedge \\ (\rho,\kappa) \models_{\mathsf{RM}} P : \psi \end{array}}{(\rho,\kappa) \models_{\mathsf{RM}} \langle E_1, \cdots, E_k \rangle .\, P : \psi}$$

(Input)

$$\frac{\begin{array}{c} \wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_i\ \wedge \\ \forall \langle V_1, \cdots, V_k \rangle \in \kappa :\ \wedge_{i=1}^{j}\ V_i \mathrel{\mathsf{E}} \vartheta_i \Rightarrow\ \wedge_{i=j+1}^{k}\ V_i \in \rho(\lfloor x_i \rfloor)\ \wedge \\ (\rho,\kappa) \models_{\mathsf{RM}} P : \psi \end{array}}{(\rho,\kappa) \models_{\mathsf{RM}} (E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k).\, P : \psi}$$

(Decryption)

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

(Asymmetric decryption)

$$\frac{\begin{array}{c} \rho \models E : \vartheta \quad \wedge \quad \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i\ \wedge \\ \forall (m^{\pm}, m^{\mp}) : \forall\ \{\!|V_1, \cdots, V_k|\!\}_{V_0}^{\ell}[\mathsf{dest}\, \mathcal{L}\,] \in \vartheta : \forall V_0' \mathrel{\mathsf{E}} \vartheta_0 : \{V_0, V_0'\} = \{m^{\pm}, m^{\mp}\} \wedge \\ \wedge_{i=1}^{j} V_i \mathrel{\mathsf{E}} \vartheta_i\ \Rightarrow\ \wedge_{i=j+1}^{k}\ V_i \in \rho(\lfloor x_i \rfloor)\ \wedge \\ (\neg \mathsf{RM}(\ell, \mathcal{L}', \ell', \mathcal{L}) \Rightarrow (\ell, \ell') \in \psi)\ \wedge \\ (\rho,\kappa) \models_{\mathsf{RM}} P : \psi \end{array}}{(\rho,\kappa) \models_{\mathsf{RM}} \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k|\!\}_{E_0}^{\ell'}\ [\mathsf{orig}\, \mathcal{L}'\,]\ \mathsf{in}\ P : \psi}$$

Table 4.6: Analysis of processes, $(\rho,\kappa) \models_{\mathsf{RM}} P : \psi$.

of the annotations leading to an abortion of the reference monitor, from the fact that the encrypted term E is decrypted at an unexpected place ($l' \notin \mathcal{L}$) or that the decrypted values where encrypted at an unexpected place ($l \notin \mathcal{L}'$), then the error component must contain the annotations where the error occurred $(l, l') \in \psi$. If the decryption succeeds the continuing process P is analysed. The only difference between the symmetric and asymmetric decryption is the requirements that the keys used are a matching pair instead of being identical. No further explanation of the asymmetric decryption should therefore be needed.

It is proved in [8] that if $(\rho, \kappa) \models_{\mathsf{RM}} P : \psi$ then the triple (ρ, κ, ψ) is a valid estimate for *all* the states passed through in an execution of P. It is also proved that when $\psi = \emptyset$ in an estimate of the form $(\rho, \kappa) \models_{\mathsf{RM}} P : \psi$ then the reference monitor cannot abort the execution of P.

4.4 The Attacker

In order to analyse protocols for vulnerabilities, what is analysed is really the LySa process in parallel with so-called Dolev-Yao attacker [14]. The attacker can perform the same operations as a normal principal, such as encryption, decryption, message send and receive etc. The result of the analysis of a process P analysed in parallel with the attacker return the least solution that satisfy the rules from the previous Section (ρ, κ, ψ). Besides the variable bindings for the process P, the analysis result also contain the variable bindings for the attacker. All possible variables of the attacker are mapped to the canonical variable $z_\bullet$, and the all names to the canonical name $n_\bullet$. The attacker is able to encrypt and decrypt terms, using his knowledge, therefore there exist annotations for the attacker, where $l_\bullet$ is a crypto-point in the attacker and $\mathcal{C}$ is the set of crypto-points in the original process P in parallel with the attacker. Also there exist a public/private key-pair belonging to the attacker $\{m_\bullet^+, m_\bullet^-\}$. Using these notations the formal definition from [8] of the attacker is given in Table 4.7 on the next page.

These conditions describe an attacker capable of carying out attacks on closed processes[2] P of the type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}^+_{\mathsf{Enc}})$. Here $\mathcal{N}_{\mathsf{f}}$ is the set of free names of the process, $\mathcal{A}_\kappa$ is the set of all arities used for sending and receiving, and $\mathcal{A}^+_{\mathsf{Enc}}$ is the set of all arities used for encryption and decryption. We now give a short description of the conditions.

1. The attacker is able to eavesdrop all messages send on the network. The values of these messages are added to the knowledge of the attacker.

2. If the attacker knows the correct key $V_0 \mathrel{\mathsf{E}} \rho(z_\bullet)$, he is albe to decrypt the term and thereby adding the decrypted values to the knowledge of the attacker $\wedge_{i=1}^{k} V_i \in \rho(z_\bullet)$. If the message was not intended to be decrypted at the attacker an error $(l, l_\bullet)$ should be added to the error component ψ telling that something encrypted at l was actualy decrypted by the attacker at $l_\bullet$.

[2] A process P is closed if it has no free variables. That is, a variable which is never bound to a name.

$$(1)\ \wedge_{k\in\mathcal{A}_\kappa} \forall\langle V_1,\cdots,V_k\rangle \in \kappa : \wedge_{i=1}^{k} V_i \in \rho(z_\bullet)$$

$$\begin{aligned}&(2)\ \wedge_{k\in\mathcal{A}^+_{\mathsf{Enc}}} \forall\{V_1,\cdots,V_k\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \rho(z_\bullet) :\\ &\quad V_0 \mathrel{\mathsf{E}} \rho(z_\bullet) \Rightarrow (\wedge_{i=1}^{k} V_i \in \rho(z_\bullet) \wedge (\neg\mathsf{RM}(\ell,\mathcal{C},\ell_\bullet,\mathcal{L}) \Rightarrow (\ell,\ell_\bullet) \in \psi))\end{aligned}$$

$$(3)\ \wedge_{k\in\mathcal{A}^+_{\mathsf{Enc}}} \forall V_0,\cdots,V_k : \wedge_{i=0}^{k} V_i \in \rho(z_\bullet) \Rightarrow \{V_1,\cdots,V_k\}^{\ell_\bullet}_{V_0}[\mathsf{dest}\,\mathcal{C}\,] \in \rho(z_\bullet)$$

$$(4)\ \wedge_{k\in\mathcal{A}_\kappa} \forall V_1,\cdots,V_k : \wedge_{i=1}^{k} V_i \in \rho(z_\bullet) \Rightarrow \langle V_1,\cdots,V_k\rangle \in \kappa$$

$$(5)\ \{n_\bullet\} \cup \lfloor\mathcal{N}_{\mathsf{f}}\rfloor \subseteq \rho(z_\bullet)$$

$$\begin{aligned}&(6)\ \forall(m^+,m^-) : \wedge_{k\in\mathcal{A}^+_{\mathsf{Enc}}} \forall\{\!|V_1,\cdots,V_k|\!\}^{\ell}_{V_0}[\mathsf{dest}\,\mathcal{L}\,] \in \rho(z_\bullet) :\\ &\quad \forall V_0' \mathrel{\mathsf{E}} \rho(z_\bullet) : \{V_0,V_0'\} = \{m^\pm,m^\mp\} \Rightarrow (\wedge_{i=1}^{k} V_i \in \rho(z_\bullet) \wedge\\ &\qquad\qquad (\neg\mathsf{RM}(\ell,\mathcal{C},\ell_\bullet,\mathcal{L}) \Rightarrow (\ell,\ell_\bullet) \in \psi))\end{aligned}$$

$$(7)\ \wedge_{k\in\mathcal{A}^+_{\mathsf{Enc}}} \forall V_0,V_1,\cdots,V_k : \wedge_{i=0}^{k} V_i \in \rho(z_\bullet) \Rightarrow \{\!|V_1,\cdots,V_k|\!\}^{\ell_\bullet}_{V_0}[\mathsf{dest}\,\mathcal{C}\,] \in \rho(z_\bullet)$$

$$(8)\ \{m^+_\bullet,m^-_\bullet\} \subseteq \rho(z_\bullet)$$

Table 4.7: Dolev-Yao condition.

3. The attacker create an encrypted term of the same length (arity) as any other encrypted terms in the process P using the values in his knowledge. This encrypted term will have the annotations that it was created at $l_\bullet$ and has the destination $\mathcal{C}$. This could lead to an error in the error component of the type $(l_\bullet, l)$ expressing that something encrypted at the attacker was uexpected decrypted by a process P at l.

4. The attacker is able to send out a messages on the network $\langle V_1,\cdots,V_k\rangle \in \kappa$ with the same length as any other messages $k \in \mathcal{A}_\kappa$ send in the process P. The messages are contructed using the knowledge of the attacker $\rho(z_\bullet)$.

5. All free names $\mathcal{N}_{\mathsf{f}}$ of the process P as well as the canonical name $n_\bullet$ must be added to the knowledge of the attacker.

6. As for condition (2) the attacker can decrypt a term encrypted using asymetric encryption, if he possesses the knowledge of the oposite key used for encryption.

7. As for condition (3) the attacker can create an encrypted term using asymetric encryption.

8. The attacker possesses a private/public keypair.

The Dolev-Yao conditions are related to the possible attacks discussed in section 2.4 on page 10 as follows; Condition (1) expresses that the attacker is able to eavesdrop any messages in the protocol, condition (2) and (6) enables the attacker to read eavesdrop message even though they are encrypted if the correct key is somehow in the knowledge (possible eavesdropped earlier). Condition (4) allows the attacker to send out messages on the network, enabling a replay of a previous eavesdropped message. Condition (3) and (7) enables the attacker to create new encryptions, combining this with the ability to send out messages the attacker described by these conditions should also be able to carry out modification and Man-In-the-Middle attacks. Deletion attacks are not expressed by this attacker since a message on the network κ cannot be removed from the network. The attacker is however capable of doing insertion attacks by creating a message and sending it out on the network. Insider attacks does not relate to how the attacker is modelled, rather how certificates are used. Real attacks usually consists of combinations of these attacks.

The soundness of the Dolev-Yao conditions is proved in [8].

4.5 LySa Tricks

Process algebras such as LySa are constructed from a point of view where *small is beautifully*, because it is much simpler to do proofs on a small algebra. Therefore the first impression of LySa could be that it is impossible to express many things using this notation, but gaining some insights on LySa helps to express even very complex process in a very compact way. We will in this section shortly describe some of the "tricks" we have used to express complex processes in LySa.

Leaking Information

Using the LySa-tool it is obvious that the result of the analysis very much depends on what information the attacker possesses. In the definition of LySa and the attacker it is not possible directly to add information to the attacker. In order to do so, condition (1) of the attacker should be examined (see Table 4.7 on the preceding page). All information sent on the network is eavesdropped by the attacker, using this condition it is possible to add information to the attacker by sending the information on the network.

$$\begin{array}{l} \langle Leaked\ Information\rangle.\mathsf{0} \\ | \\ P \end{array}$$

Hash-functions in LySa

In modern protocols hash-functions are used to guarantee data integrity. Hash functions are defined such that the hash-function applied on any message should give a hash-value $Hash(Mess) = HashValue$ such that two hash-values are only equal if the hash-function is applied to the same message: $Hash(Mess_1) = Hash(Mess_2) \Leftrightarrow Mess_1 = Mess_2$. By definition there does not exist an inverse hash-function e.g. it is impossible to calculate the original message from the hash-values.

To model such a function i LySa we consider the constructs used for asymmetric encryption and decryption. It is only possible to decrypt a term encrypted using asymmetric encryption if there exist a matching key to the one used to encrypt the term. Such a matching key require that the term was encrypted using a key from a valid key pair created using the $(\nu_\pm\ K)$ construct. If we instead of such a key uses a free name say $Hash$:

$$\{|Mess|\}_{Hash}$$

It is impossible to decrypt the message since $Hash$ is not a key in a key-pair. On the other hand we can still do matching since the free name $Hash$ can be used in all processes to created encrypted term of the same form, if the message $Mess$ is known.

Variable bindings in LySa

When complex keys and hash-functions are used in the encoding of a protocol in LySa it would be nice to have an assignment construct enabling you to store large terms in a temporary variable, and the just matching the variable to another term.

Using the decrypt term it is possible to archive the same effect as if assignments where a part of the LySa-calculus. If a protocol uses a term $\{N_1, \{|N_1|\}_{Hash}|\}_K$ as a complex session key, it would be nice to store this term in a variable, and then use this variable for decryption and encryption. This could be done as:

$$\mathsf{decrypt}\ \{\{N_1, \{|N_1|\}_{Hash}|\}_K\}_{ASSIGN}\ \mathsf{as}\ \{;\, x_{Key}\}_{ASSIGN}$$

First the entire term (the session key) is encrypted using a free name $ASSIGN$, thereafter the term is decrypted using the same name at a place, where no matching takes place, only a binding to the variable so that the complex key should be in the estimate for the variable $\{N_1, \{|N_1|\}_{Hash}|\}_K \in \rho(x_{Key})$.

Certificates in LySa

If certificates are used in a protocol described by a LySa-process (e.g. the TLS protocol uses certificates), a PKI must be part of the LySa-process. Certificates are signed by a Certificate Authority (CA), to be able to do so there must exist a public/private key pair of such an authority, and to ensure that the attacker does not know the private key, the keys should be restricted $(\nu_\pm\ CA)$. Now each of the principals described in the LySa-process could create it own public/private key-pair, and then use the private key of the CA CA^- to sign its own certificate of the form: $\{|C, K_C^+|\}_{CA^-}$ where C is the name of the principal, and K_c^+ is the public key belonging to the principal. In this way all other principals are able to verify the certificate using the public key of the CA to decrypt the signed message. The attacker is however not able to create a certificate since the public/private key pair of the CA is restricted. If the keys where not restricted the attacker could create a certificate for every principal, since the names of the principals are free names in LySa-processes, and thereby impersonate all principals to the others[3]. To give the attacker a certificate without enabling him to do insider-attacks a signed certificate is leaked on the network together with the public key of the CA CA^+. LySa-processes using certificates have the following form:

$$\begin{array}{l}(\nu_\pm\ CA)\,(\\ \langle CA^+, \{\!|n_\bullet, m_\bullet^+|\!\}_{CA^-}\rangle.0\\ |\\ \ldots\\ )\end{array}$$

The certificate of the attacker $\{\!|n_\bullet, m_\bullet^+|\!\}_{CA^-}$ contain the name of the attacker $n_\bullet$ and the public key of the attacker $m_\bullet$. The can also verify other certificates using the leaked key CA^+.

[3]This is an insider-attack

4.6 MetaLySa

So far we have seen LySa and the analysis which can be used upon it. Now we shall look at how protocols are modelled in LySa and introduce a way to handle many parallel instances of the same process.

MetaLySa is an extension or a layer on top of LySa in order to conveniently model several parallel instances of the same protocol. This idea is to add indices on names and variables (eg. x_i). An index can be instantiated as a number in a range of numbers (eg. $[0; 2]$). When indices are instantiated MetaLySa is converted to LySa see Figure 4.2. Nothing conceptually new is added by using MetaLySa, only convenient abbreviations are included.

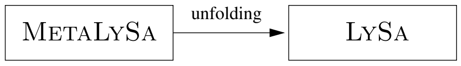


Figure 4.2: MetaLySa & LySa

New Constructs

MetaLySa contains a construct used to create several parallel instances of the same protocol. The construct is $|_{i=a}^{n}$ where i is an index descriptor and a is the first instantiation of the index and n is the maximum value of an index. n is chosen globally, meaning it is not possible to have $|_{i=1}^{n}$ and $|_{j=0}^{n'}$ where $n \neq n'$. For example the construct $|_{i=1}^{n}|_{j=0}^{n} \; P_{i,j}$ is an abbreviation for:

$$P_{1,0}|P_{1,1}|\ldots|P_{1,n}|\ldots|P_{n,0}|P_{n,1}|\ldots|P_{n,n}$$

Here $P_{1,0}$ is a notation for the process P where names and variables with indices i and j replaced with 1 and 0 respectively. The n which is the upper bound for indices is determined when MetaLySa is expanded to LySa before using it as input to the analysis. Usually indices start from 1 and depending on the value chosen for n the expansion will result in more or less processes. It is also possible to use another form this parallel construct which allow an exception to the values of the index. In the example above indices were in the range $[a; n]$. It is possible to make an exception to this removing one index from this range using $|_{i=a\backslash j}^{n}$. This mean the index i can take values in the range $[a; n]$ except the value if the index j. That is, i cannot have the same value as j. We can change the example from before to $|_{i=1}^{n}|_{j=0\backslash i}^{n} \; P_{i,j}$. Compared with the example from above this would remove the processes: $P_{1,1}, \ldots, P_{n,n}$.

MetaLySa also has a construct for indexed restrictions defined by :

$$(\nu_{i=a_1,j=a_2,\ldots}^{n} \; name_{i,j,\ldots})$$

Unlike the parallel construct being an abbreviation for parallel processes this is an abbreviation for a sequence of restrictions. Also all indices must be listed in this construct so indices i, j, etc. is listed in the construct. As an example $(\nu_{i=1,k=1}^{n} \; name_{i,k})$ is an abbreviation for:

$$(\nu \, name_{1,1}) \, (\nu \, name_{1,2}) \, \ldots (\nu \, name_{1,n}) \, \ldots (\nu \, name_{n,n})$$

A similar construct exist for restricting names for asymmetric keys and has the form: $(\nu^n_{i=a_1,j=a_2,\ldots}\ \pm\ name_{i,j,\ldots})$.

Structure

When modelling protocols using LySa the structure used is the same regardless of what specific protocol is being modelled. In a protocol a principal can play one or more roles. The roles might be Client, Server, Certificate Authority, etc. The structure of the model in LySa is the same only the interpretation changes. Many protocols also require certain secrets to be in place before the protocol can begin. This could be a shared secret or a certificate. A model of this in LySa first restricts the names that are secret to the attacker. Then the process is split into parallel processes, one for each role in the protocol.

For example the Wide-Mouth-Frog protocol. It has an Initiator, a Responder and a Key-Server. Each principal has a shared key with the Key-Server. A principal can be both an Initiator and a Responder at the same time but only one Server exists and it is distinct form the others. This is modelled as:

$$\begin{array}{l}(\nu\, K)\\ \quad Initiator\\ |\\ \quad Responder\\ |\\ \quad Server\end{array}$$

Index Value Zero

The index $i = 0$ is usually reserved for the attacker. There is nothing special about index 0 but in some scenarios it is convenient to use this index for roles the attacker can play. For example a scenario where a two principals A and B use a shared secret key K to communicate with each other. If A and B are two different types of roles, meaning that one principal cannot act as both A and B, A could be indexed with i and B with j. In MetaLySa this would looks like:

$$\begin{array}{l}(\nu^n_{i=1,j=1} K_{i,j})\\ |^n_{i=1}\, |^n_{j=1}\ A\\ |\\ |^n_{j=1}\, |^n_{i=0}\ B\end{array}$$

In this scenario we have i processes of type A and j processes of type B. Each pair of A_i and B_i share a secret key $K_{i,j}$ and this is used to communicate with each other. Notice that in this example we allow the index i in the process B to be in the interval $[0; n]$. We say that this is the same as allowing B to talk to the attacker. Notice that the key $K_{i,j}$ is only restricted with both indices: $i, j \in [1; n]$. This means that when the process B indexed with $i = 0$ tries to communicate it will do so with a key that is not restricted, hence known by the attacker. In the conversion from MetaLySa to LySa the annotations on encryptions and decryptions are changed. If a destination or origin set of crypto-points include a crypto-point with an index of 0 the crypto-point of the attacker $\ell_\bullet$ is included in the set. This ensures that none of the crypto-points where the process is talking directly with the attacker (modelled by a process with index = 0) is included in the ψ component of the result.

The scenario is a bit more complicated when using asymmetric keys. These keys can occur in the process in three forms. As either minus or plus name or as a name in a restriction declaring an asymmetric key pair. If the restriction occur in a process both the plus and the minus key is added to the space of names in the process. If on the on the contrary only the plus key or the minus key occur in a process the matching key partner is not added to the space of names. This could be a problem when using scenarios where some roles should talk to the attacker. For example consider this LYSA process:

$$\begin{array}{l} (\nu\, Mess)\,(\\ \langle \{\!|Mess|\!\}_{k^-} \rangle. \\ 0 \\ ) \end{array}$$

Here the name $Mess$ is restricted and thereafter sent on the network asymmetric encrypted with the unrestricted key k^-. The key k^- is known by the attacker but since the key k^+ does not exist in the process, the attacker cannot decrypt the message and read $Mess$. On the other hand if the process is changed to:

$$\begin{array}{l} (\nu\, Mess)\,(\\ \langle \{\!|Mess, k^+|\!\}_{k^-} \rangle. \\ 0 \\ ) \end{array}$$

The attacker will now be able to decrypt the message and read $Mess$. In our setting this means we have to ensure that for all asymmetric key both parts of the key exist in the process.

A similar problem occur when dealing with certificates of other complex shared key structures. A certificate is modelled by $\{identity, public - key^+\}_{CA^-}$. We restrict the key CA^-, which restricts the attacker from issuing certificates. The attacker still needs a certificate if he should interact with any of the roles using a certificate. If we have a role A and it is indexed to A_i. To allow the attacker to act as A_0 we must supply him with the certificate $\{A_0, key^+\}_{CA^-}$ and the key key^-.

Value of n

We have now seen how METALYSA can be used to create many parallel processes or many restricted names. This was done using indices in an interval. The lower bound of the interval can be given at each place where a METALYSA construct is used but the upper bound n is set globally. The question is, what the value of n is, where increasing n would not allow the analysis to find more errors.

Let us consider a scenario where we have an infinite number of parallel processes. We would have the Dolev-Yao attacker $\mathcal{A}$ in parallel with the processes P_x with $x \in \mathbb{N}$:

$$\mathcal{A}|P_1|P_2|\cdots|P_j|\cdots|P_k|\cdots$$

If the protocol modelled has two roles each P would be two parallel processes. This means that a process P is one instance of a protocol. The errors found by the analysis originate form the annotations of encryptions and decryptions with crypto-points and a set of either destination crypto-points or origin crypto-points. An error is found if an encrypted value is decrypted at a crypto-point not in the destination set or if the value was not encrypted at a crypto-point in the origin set. For errors where a value is either encrypted or decrypted by the attacker resulting in an error only one instance of a protocol is needed. Errors could also occur within a single instance of a protocol. That is, a message from one step of the protocol could end up in another step of the protocol. This would also only require one instance of the protocol. Generally an encrypted value would originate from one process P_j and be decrypted at another process P_k in which case we need two instances. This gives us that any error would be found if $n = 2$. This does, though, not mean that we can find any error in a protocol but merely that if the analysis is able find an error in a scenario with n parallel processes it would also find it if $n = 2$.

4.7 Using the Analysis

We have used the implementation of the LySa-tool from [38]. The flow of the analysis is illustrated on Figure 4.3. The LySa-tool parses a LySa process from an ASCII-file then transforms it to ALFP logic equations which are solved by the Succinct Solver [32]. The Succinct Solver[4] is a tool for solving constraints specified in ALFP, we shall not go further into these details here. The Succinct Solver computes the minimum solution satisfying the input equations and returns a result. This result from the Succinct Solver is transformed by the LySa-tool, to a readable version of the estimate (ρ, κ, ψ). The LySa-tool also

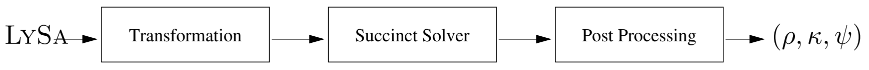


Figure 4.3: Implementation of the analysis

implements pretty-print functions for the LySa processes, and analysis results, this is very useful for debugging and interpretation of the analysis result.

Analysis of WMF

To use the analysis on the WMF protocol the LySa-tool takes an ASCII version of the LySa-process presented in Table 4.3 on page 40, this ASCII implementation of LySaprocess for the WMF protocol can be found in Appendix A on page 99. The analysis result gives an estimate where $\psi = \emptyset$. The variable environment ρ is given below, where $V_\bullet$ denotes any value the attacker has the knowledge of e.g. $V_\bullet \in \rho(z_\bullet)$.

[4]An implementation can be found at [39].

$$\boxed{\begin{array}{rcl}
\rho(xB) & = & V_\bullet, B \\
\rho(xKey) & = & K \\
\rho(xMess) & = & V_\bullet, \{K\}_{K_A} \\
\rho(yA) & = & V_\bullet, A \\
\rho(yKey) & = & K \\
\rho(yMess) & = & V_\bullet, \{K\}_{K_A} \\
\rho(yMessages) & = & V_\bullet, \{Secret\}_K \\
\rho(ySecret) & = & Secret \\
K & \notin & rho(z_\bullet) \\
Secret & \notin & rho(z_\bullet)
\end{array}}$$

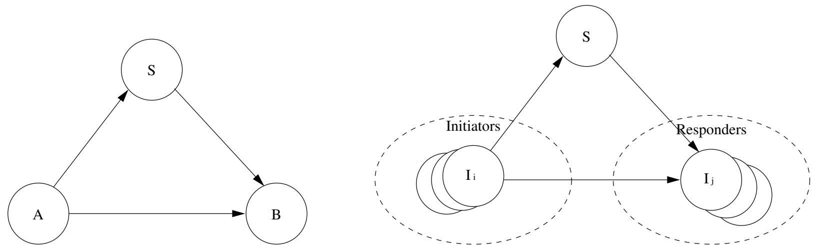
The κ component is left out since the over-approximation of the messages on the network is of no interest for the properties such as authenticity and confidentiality we are interested in.

The case that $\psi = \emptyset$ ensures that there does not occur encryptions/decryptions at unexpected places, therefore it should not come as a surprise that the attacker does not possess the knowledge of the *Secret*. All variables bound to values received directly[5] on the network can all be bound to anything inside the attacker $V_\bullet$. This is because the attacker can send out a message of any length matching any term receiving values, and the names the process matches on are all free names so the attacker has the knowledge to create messages on the right form. The attacker does however not possess information enough needed to create encrypted messages to be mistaken for messages en the correct run of the protocol (this would lead to an error in ψ of the form $(l_\bullet, l)$, where l is any point in the LySa-process), neither is the attacker able to decrypt any of the encrypted terms from the message on the network.

Now one could think that the WMF protocol was verified to guarantee authenticity and confidentiality, since messages believed to originate from A actually do so, and messages to be kept secret are not leaked to the attacker. This is however only the case in this very specific scenario described in the LySa-process. The LySa-process from Table 4.3 on page 40 describes a fixed scenario, where there only exist one initiator A, one server S and one responder B this scenario is shown on Figure 4.4(a). This scenario also limits the attacker to act as a passive attacker; since there only exist one initiator and one responder, the attacker cannot act at either of them, this restricts the attacker only to eavesdrop messages sent in the protocol, and use these messages to send out new messages.

What we want to analyse is a more flexible scenario, where there exist a number of initiators I_i, and a number of responders I_j, such a scenario is depicted on Figure 4.4(b). In this more flexible scenario the attacker is able to act as either an initiator or as a responder in a protocol run, since all initiators and responders share keys with the server the attacker also shares keys with the server S. To describe a more flexible scenario we use MetaLySa-extension described in section 4.6 on page 51. The LySa-process in Table 4.8 on page 57 describes the more flexible scenario. Initially shared keys KA_i and KB_j are restricted for the valid principals $1 \leq i, j \leq n$. The keys belonging to the attacker KA_0 and KB_0 are not restricted and are therefore threaded as free names in the analysis. The LySa-description of the initiating principal I_i is indexed from $i = 1$ and $j = 1$ because we only describe the legitimate part of the system. The server S

[5]Directly in the sense they are not decrypted from values received on the network



(a) Fixed WMF scenario (b) Flexible WMF scenario

Figure 4.4: Different WMF scenarios

is indexed from $i = 0$ and $j = 0$ allowing the attacker to act as either a initiator or a responder in the protocol. The responder I_j is indexed from $i = 0$ and $j = 1$ allowing the responder to actually receive messages from the attacker. All indices are restricted $i \neq j$ to avoid principals to try to authenticate themselves.

Using this LySa-process we run the analysis again, and now we get a entirely different result:

$$\begin{array}{|rcl|}
\hline
\psi & = & (A2_{i,j}, l_\bullet), (l_\bullet, B2_{i,j}), (A1_{1,2}, S1_{2,1}), \ldots, (A2_{1,2}, B2_{2,1}), \ldots \\
\hline
\rho(ySecret_{1,2}) & = & V_\bullet, Secret_{1,2}, Secret_{2,1}, \ldots \\
\rho(ySecret_{2,1}) & = & V_\bullet, Secret_{2,1}, Secret_{1,2}, \ldots \\
 & \vdots & \\
K_{i,j} & \in & rho(z_\bullet) \\
Secret_{i,j} & \in & rho(z_\bullet) \\
\hline
\end{array}$$

The result $\psi \neq \emptyset$ shows that encryptions and decryptions have occurred at places not expected. Components of the type $(A2_{i,j}, l_\bullet)$ shows that information encrypted at $A2$ in any principal initiating the protocol can be decrypted by the attacker. $A2$ is the place where the $Secret_{i,j}$ is encrypted and therefore this pair in the error-component implies that all secrets are known by the attacker $Secret_{i,j} \in rho(z_\bullet)$. Components of the type $(l_\bullet, B2_{i,j})$ shows that the attacker is able to impersonate any initiator of the protocol to the responder, and hereby sending encrypted information to the responder believed to be from the actual initiator. Using this method the attacker can send any information in his knowledge $V_\bullet$ to any responder I_j who believes the information originating from the initiator I_i, by this the variable storing this information contains values from the attacker $V_\bullet \in \rho(ySecret_{i,j})$. Errors of the types $(A1_{1,2}, S1_{2,1})$ and $(A2_{1,2}, B2_{2,1})$ indicates that messages encrypted in one run of the protocol can be decrypted in another run of the protocol, which results in the a secret meant for principal I_1 could end up at principal I_2 e.g. $Secret_{2,1} \in \rho(ySecret_{1,2})$. The result for WMF contain more errors than described here but they are of the same types as has already been described.

As mentioned in section in Section 4.3 on page 43 the analysis result is an over-approximation, this means that $\psi \neq \emptyset$ does not necessarily imply that there exist an error. The value in the error-component ψ could come from a trace

$(\nu^n_{i=1\ldots n}\ KA_i)$
$(\nu^n_{j=1\ldots n}\ KB_j)$
$|^n_{i=1\ldots n}\ |^n_{j=1\ldots n\, j\neq i}$
$!(\nu\, K_{ij})$
$\langle I_i, S, I_i, I_j, \{K_{ij}\}_{KA_i}[\text{at } A1_{ij} \text{ dest } S1_{ij}]\rangle.$
$(\nu\ Secret_{ij})$
$\langle I_i, I_j, \{Secret_{ij}\}_{K_{ij}}[\text{at } A2_{ij} \text{ dest } B2_{ij}]\rangle.$
0
$|$
$|^n_{i=0\ldots n}\ |^n_{j=0\ldots n\, j\neq i}$
$!(I_i, S, I_i;\ xB_{ij}, xMess_{ij}).$
decrypt $xMess_{ij}$ as $\{;\ xKey_{ij}\}_{KA_i}[\text{at } S1_{ij} \text{ orig } A1_{ij}]$ in
$\langle S, xB_{ij}, I_i, \{xKey_{ij}\}_{KB_j}[\text{at } S2_{ij} \text{ dest } B1_{ij}]\rangle.$
0

$|$
$|^n_{j=1\ldots n}\ |^n_{i=0\ldots n\, i\neq j}$
$!(S, I_j;\ yA_{ij}, yMess_{ij}).$
decrypt $yMess_{ij}$ as $\{;\ yKey_{ij}\}_{KB_j}[\text{at } B1_{ij} \text{ orig } S2_{ij}]$ in
$(yA_{ij}, I_j;\ yMessages_{ij}).$
decrypt $yMessages_{ij}$ as $\{;\ ySecret_{ij}\}_{yKey_{ij}}[\text{at } B2_{ij} \text{ orig } A2_{ij}]$ in
0

Table 4.8: Flexible WMF scenario specified in MetaLySa

that is actual not possible. To conclude anything about a protocol analysed to have errors one have to use the analysis result to find an actual trace leading to an this error. In the following example $\rightarrow\ M(S)$ denotes that the attacker is able to eavesdrop the message intended for principal S, and $M(I_1)\ \rightarrow$ denotes that the attacker impersonates the principal I_1.

1.	I_1	$\rightarrow$	$M(S)$	:	$I_1, I_2, \{K_{1,2}\}_{KA_1}$
1.′	$M(I_1)$	$\rightarrow$	S	:	$I_1, I_0, \{K_{1,2}\}_{KA_1}$
2.	S	$\rightarrow$	I_0	:	$I_1, \{K_{1,2}\}_{KB_0}$
3.	I_1	$\rightarrow$	$M(I_2)$	:	$\{Secret\}_{K_{1,2}}$

The trace above of the flexible WMF-protocol leads to the error $(A2_{1,2}, l_\bullet)$ in 4 steps.

1. The initiating principal I_1 wants to engage communication with a responding principal I_2 using the key $K_{1,2}$ as session key. The newly created session key is encrypted using the shared key KA_1. The attacker is able to eavesdrop the message intended for the server S

1.' The attacker now changes the message so that it impersonates I_1 who wants to engage communication with I_0.

2. The server S recognises the message as a message from the initiating principal I_1 who wants to communicate with the principal I_0 (who is in fact the attacker) therefore the session key is send to I_0 using the shared key KB_0.

3. The attacker is able to decrypt the session key, and can now decrypt any messages send from I_1 to I_2, indicated by $(A2_{1,2}, l_\bullet)$.

Using the session key $K_{1,2}$ learned above, the attacker is able impersonate I_1 to the responder I_2:

$$\begin{array}{llcll}
1. & M(I_1) & \rightarrow & M(S) & : & I_1, I_2, \{K_{1,2}\}_{KA_1} \\
2. & S & \rightarrow & I_2 & : & I_1, \{K_{1,2}\}_{KB_2} \\
3. & M(I_1) & \rightarrow & I_2 & : & \{FalseSecret\}_{K_{1,2}}
\end{array}$$

1. The first message from the previous run of the protocol is replayed by the attacker, leading the Server S to believe that I_1 has created at fresh key for communication with I_2.

2. The session key is sent to I_2, who believes it to be shared with I_1, but in fact it is shared with the attacker.

3. The attacker is now able to impersonate I_1, by sending false secrets to I_2

This protocol run leads to the error $(l_\bullet, B2_{1,2})$ because the decrypted secret was believed to originate from the initiator I_1 but was in fact encrypted at the attacker.

The third type of error, where the error component indicate that encryption/decryption inside the real protocol could lead to errors e.g. $(A1_{1,2}, S1_{2,1})$ is not discussed here. Here the WMF protocol is only used to explain how the analysis is used. The analysis result of the WMF protocol is thoroughly discussed in [8] therefore we shall leave out further discussion of WMF.

4.8 Limitations of the Analysis

As mentioned in Section 4.3 on page 43 the analysis returns an over-approximation of the variable environment ρ and network component κ, the error-component ψ is therefore also an over-approximation because the reference monitor could abort due to the over-approximation of the two other components. In this section we describe some the details that could lead either to an over-approximation of the analysis, or a misunderstanding of the analysis result.

4.8.1 Analysis with and without the Attacker

When using the LySA-tool it is important to be aware that a result from the analysis on a LySA-process P in parallel with the Dolev-Yao attacker $\mathcal{A}$, is not the same as a result where P is analysed alone.

$$(\rho, \kappa) \models_{\mathsf{RM}} P|\mathcal{A} : \psi \quad \neq \quad (\rho', \kappa') \models_{\mathsf{RM}} P : \psi'$$

All communications occurring in P can also occur in $P|\mathcal{A}$, therefore the estimates for the process P alone is a subset of the estimates for the process in parallel with the attacker e.g.

$$\forall x \in \mathcal{X}\ \rho'(x) \subseteq \rho(x)\ \wedge\ \kappa' \subseteq \kappa\ \wedge\ \psi' \subseteq \psi$$

If the variable environment ρ is considered to verify that certain variable bindings occur, one must be aware that it might only be the case in the presence of an attacker. To assure that the variable bindings also occur without the presence of the attacker the variable environment ρ' from the analysis without

$(\nu\, K)$ (
$(\nu\, \mathit{Secret})$
$\langle A, C, \{\mathit{Secret}\}_K[\mathsf{at}\, A\ \mathsf{dest}\, B\,]\rangle.$
$\mathsf{0}$
$|$
$(A, B;\ \mathit{xMess}).$
$\mathsf{decrypt}\ \mathit{xMess}\ \mathsf{as}\ \{;\ \mathit{xSecret}\}_K[\mathsf{at}\, B\ \mathsf{orig}\, A\,]\ \mathsf{in}$
$\mathsf{0}$
)

Table 4.9: Influence of the Attacker

the attacker should be considered. An example showing this is presented in Table 4.9. There exists a long term key K between two principals A and B. A wishes to communicate a secret to B, but there has been made an error in the implementation of the LySa-process, so A sends the secret to a principal C: $\langle A, C, \{\mathit{Secret}\}_K[\mathsf{at}\, A\ \mathsf{dest}\, B\,]\rangle$. The result of the analysis with and without the attacker yields the following result:

Without the Attacker		With the Attacker
$\psi' = \emptyset$	$\subseteq$	$\psi = \emptyset$
$\rho'(xSecret) = \emptyset$	$\subseteq$	$\rho(xSecret) = Secret$

In the result of the analysis with the presence of an attacker, the message is in fact communicated to the principal B. This is actually done by the attacker, who is carrying out a replay attack on B, replaying a modified version of the message sent from A to C matching the input in B so the principal B believes it is originating from A. When the message is received B is able to decrypt the $Secret$ using the long term key K and binding it to the variable $xSecret$. If an attacker did not exist, the message could not be modified and therefore the message would not match the input pattern of B. The semantics of LySa requires input patterns to match the values of the output, to do a communication, since this is not the case the communication never occurs when there does not exist an attacker. Since B never receives the encrypted $Secret$ it is impossible to decrypt and therefore no value is bound to $xSecret$.

4.8.2 Independent Attribute Analysis

The analysis implement a *Independent Attribute Analysis* [31]. This means that terms and their evaluations are treated independently even if a strong relation between them exists. Consider the example presented in Table 4.10 on the following page. A principal A wants to send an encrypted nonce N to the principal B using the newly created key K, alongside the encrypted nonce the free name any is sent. After this the principal A once more sends any and afterwards the key K used to encrypt the nonce. The principal B receives the values over the network and binds them to the variables yA and yK, the encrypted term is then decrypted and bound to the variable yS. In this LySa-process one input term in principal B: $(A, B;\ yA, yK)$ is used to receive both

```
$(\nu\, N)\,(\nu\, K)$
$\langle A, B, \{N\}_K, any\rangle.$
$\langle A, B, any, K\rangle.$
0
|
$(A, B;\ yA, yK).$
decrypt $yA$ as $\{;\ yS\}_{yK}$ in
0
```

Table 4.10: Example of the independent attribute

outputs from principal A. Running the analysis we get following result[6]:

$\rho(yA)$	$=$	$\{N\}_K, any$
$\rho(yK)$	$=$	K, any
$\rho(yS)$	$=$	N

The variables yA and yK in principal B both have the values corresponding to the output from principal A, and the value of the variable $\rho(yS) = N$ indicates that B is able to decrypt the encrypted nonce N. If we take a closer look at the variables yA and yK, they are both bound at the same place e.g. in the term receiving values a message from A: $(A, B;\ yA, yK)$. Since the variables are bound at the same place, there exist a relation between the values bound to them. If the values originate from the first output in A the variables would be bound as $\{yA, yK\} = \{\{N\}_K, any\}$, after receiving the second message from A the values would be bound as $\{yA, yK\} = \{any, K\}$. Considering this the decryption decrypt yA as $\{;\ yS\}_{yK}$ should never take place since this require the variables to be bound to $\{yA, yK\} = \{\{N\}_K, K\}$ which can never happen. This is however not captured by the analysis, since there does not exist any bindings between variables, there only exist a set of possible values for each variable. When the decryption is analysed the analysis checks whether there exist an encrypted term in the possible values of yA, since this is the case: $\{N\}_K \in \rho(yA)$, the analysis now checks whether the correct key is in the possible values of yK and when this is verified $K \in \rho(yK)$ the nonce N is bound to yS.

This feature could of cause lead to errors in the error-component not actually possible (false positives) e.g. if the encrypted nonce was annotated with a destination different from the one at the place where the decryption occurs, an error would be reported in ψ but this error could never occur in a real implementation, e.g. the reference monitor aborts because the analysis does not capture the relation between the variables yA and yK.

4.8.3 Overall Properties

The LySa-tool is very useful to describe and analyse certain properties of cryptographic protocol, where as others are not analysed. We shall shortly describe some of the properties that are not grasped by the analysis.

[6]The ψ component is ignored as well as values originating from the attacker $V_\bullet$

No notion of time exist in LySa, therefore time stamps cannot be used. Many cryptographic protocol however use time stamps more as a nonce, since they do not rely on different clocks to be synchronised. Such protocols can be modelled in LySa using a restricted (thereby *fresh*) value instead of a time stamp.

LySa models perfect encryption which implies it is impossible to derive any encryption key from large amounts of cipher text. Leaks of information (e.g. an old session key) can be used to model the effect of a crypto analysis attacks.

Properties of liveliness cannot be verified using the LySa-tool. E.g. a denial of service attacks could never occur be found using the LySa-tool. Seen from the analysis point of view a communication can either occur since it maintain the semantics of a LySa process, or it can not. The attacker has no possibility to "block" communications by repeatedly sending out messages on the network.

Understanding the limitations of the LySa-tool, it is also important to understand, what properties actually are possible to verify. The most important ones are:

Confidentiality It is possible to verify whether a Dolev-Yao attacker or other principals are able to read confidential messages by investigating the variable environment ρ.

Authenticity Using the origin/destination annotations it is possible to verify authenticity in a protocol. Violations will be reported in the error component ψ.

Integrity The LySa-tool models perfect encryption, this can be used to guarantee data integrity. E.g. an encrypted message can only be modified, if it first is decrypted using the correct key, and then encrypted again.

4.9 SAML SSO in LySa

The protocol narration of the SAML Single Sign-On from Table 3.5 on page 23 is now reformulated as a LySa process. This process is shown in Table 4.11. We have added three indices to all variables and crypto-points, since each of the three roles can be played by arbitrarily many principals. The case where $i = 1$, $j = 2$ and $k = 3$ represents the scenario where $User_1$ requests a service at $Destination_2$ using $Source_3$ for authentication. The reason for only adding one index to each of the names identifying the principals is that the nature of a process does not change when another source-site is used for authentication or another destination-site is requested for a service. However, the variable bindings certainly do change and in order to aid the analysis we shall use three indices for the variables.

As discussed in section 3.5 on page 34 the TLS-protocol should be inserted into the SAML-protocol, we shall however first describe how a transport layer protocol is described and used in LySa in Chapter 5 on page 63.

$(\nu_{i=1}^{n}\ KU_i)$
$|_{i=1}^{n}\ |_{j=1}^{n}\ |_{k=1}^{n}$
$!\langle U_i, S_j, D_k\rangle.$
$(S_j, U_i, D_k;\ yArtifact_{ijk}).$
$\langle U_i, D_k, S_j, yArtifact_{ijk}\rangle.$
$(D_k, U_i;\ ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}[\mathsf{at}\, U_{ijk}\ \mathsf{orig}\, D_{ijk}\,]$ in
0
$|$
$|_{i=1}^{n}\ |_{j=1}^{n}\ |_{k=1}^{n}$
$!(U_i, S_j;\ xD_{ijk}).$
$(\nu\ Artifact_{ijk})\ \langle S_j, U_i, xD_{ijk}, Artifact_{ijk}\rangle.$
$(D_k, S_j, Artifact_{ijk};\ ).$
$\langle S_j, D_k, U_i, KU_i\rangle.$
0
$|$
$|_{i=1}^{n}\ |_{j=1}^{n}\ |_{k=1}^{n}$
$!(U_i, D_k;\ zS_{ijk}, zArtifact_{ijk}).$
$\langle D_k, zS_{ijk}, zArtifact_{ijk}\rangle.$
$(zS_{ijk}, D_k, U_i;\ zKU_{ijk}).$
$(\nu\ Mess)\ \langle D_k, U_i, \{Mess\}_{zKU_{ijk}}[\mathsf{at}\, D_{ijk}\ \mathsf{dest}\, U_{ijk}\,]\rangle.$
0

Table 4.11: SAML SSO in LySa

Chapter 5

CLySa

As models of protocols get more accurate the complexity of the models increase. When the complexity of a protocol increases so does the complexity of the corresponding model along with the risk of errors, also the more complex the model is the harder it is to read. This is particularly true when modelling a protocol utilising a transport layer protocol. A protocol with only a few messages can grow dramatically a transport layer protocol is used, as each message is sent with as many messages as used in the transport layer. A transport layer protocol is often used several times, and thereby instantiated several times, leaving great parts of the model almost identical, this makes the model hard to write and later read.

We have found these problems to be significant when modelling SAML SSO on top of TLS. The handshake of TLS is four messages and three instances are needed in SAML SSO which in itself are six messages. Additionally it would be interesting to test SAML SSO in scenarios with different combinations of unilateral and bilateral TLS. This all lead to the conclusion that in order to handle the processes in LySa a simple way of modelling a protocol on top of a transport layer protocol was needed. Looking at the SPI [3] calculus, it defines a notion of channels. An established connection using a transport layer protocol does in some sense act as a private channel. The idea is to use the syntax from SPI but expand channels to the LySa process elements modelling the transport layer protocol. This idea enables us to change all instances of a transport layer in a model just by changing the definition, as well as one implementation of a transport layer easily could be replaced by another. Using an expansion of channels into LySa-process we are able to use the analysis defined in [8] and explained in Section 4.3 on page 43 without any modifications.

One could ask why it is necessary to model transport layers. It could seem logical to assume certain security properties of the transport layer protocol. We are however not aware of any result showing that TLS cannot in some way interfere with itself or with the data it caries. It would also require a new analysis as each message would need some annotation of the properties provided by the transport layer.

A transport layer protocol is like any other layer in the OSI reference model. The layer may need some initial and ending messages but besides this, interfacing with the layer is simply sending and receiving messages. When we look at this layer concept in a process calculus it can be viewed as channels. The extension described in this chapter is named CLySa for “LySa with channels”. As mentioned before we do not want channels in the way SPI defines them, rather a channel is a shorthand for a LySa process modelling the current process on top of a transport layer protocol. In this sense channels are parameterised macros to be expanded into a LySa-process.

5.1 Channels

A protocol narration has a set of roles, this could be client and server. Each role is a separate principal on a network meaning that they can run independently of each other communication only by sending messages on the network. In LySa this is modelled by processes combined by parallel constructs. In a scenario with one client and one server the model would have the structure of Table 5.1.

Table 5.1: Example of a simple process

Mess#	**CLySa**
1	$Client\ldots$
	$\vert$
$1'$	$Server\ldots$

This model show a strategy where parallel constructs are only used to combine different roles in the protocol. That is, a process will split into several parallel processes first, not first send/receive a few messages and then split into more parallel processes.

What we are looking for here is a way to transform a LySa process using channels to a normal LySa process. This can be illustrated by a simple example shown in Table 5.2. Suppose we have a protocol with two roles: A, B. Only one message exchange occurs. A sends the message $\{A, B, C, D, E\}$ to B over the channel *TL*. This message is received at B, using the channel, A, B, C are matched and D, E are bound to the variables x, y. This example includes square brackets around some of the message elements, this is list construct, whose purpose and meaning will be explained later. For now this can be ignored.

Table 5.2: Example of a simple process using channels

Mess#	**CLySa**
1	$TLs\langle A, B, [C, D, E]\rangle. 0$
	$\vert$
$1'$	$TLr(A, B, C; [x, y]). 0$

Now we have to define the channel *TL*, TLs is for sending and TLr is for receiving. In this example TL is a transport layer encrypting all messages using the symmetric key K. Using this definition we can expand the process using channels to one not using channels shown in Table 5.3 on the next page.

Table 5.3: Expansion of a process using channels

Mess#	**CLySa**
1	$\langle A, B, \{C, D, E\}_K^{Cli}$ [dest $Serv$] $\rangle.\,0$
	$\mid$
$1'$	$(A, B; enc)$.
$1''$	decrypt enc as $\{C; x, y\}_K^{Serv}$ [orig Cli] in 0

In the expanded process the message elements have been encrypted. The A and B elements are not encrypted because, as said before, they act as sender and receiver addresses of the messages. In message 1 C, D, E are encrypted using the key K and furthermore this encryption is annotated with a label Cli and a destination crypto-point $Serv$. At message 1', a new variable is introduced to hold the encrypted values. These are decrypted at 1", and here C is matched while D, E are bound to x, y.

This example should give an idea of how the use of channels can be used to simplify LySa processes. Also if the transport layer modelling the channel TL is changed, there is no need to change the un-expanded LySa process. The example is however not complete, as the channel TL has not yet been defined formally. When expanding LySa processes a channel is expanded as a macro taking some arguments. A macro can be defined as:

$$TLs(C, S, M) :=< C, S, \{M\}_K^{Cli} \text{ [dest } Serv] \;\; > .0$$

The sending macro TLs has the formal parameters C, S, M. In the example from Table 5.2 on the facing page A is bound to C and B is bound to S. The bindings to M is more complicated. M is the message payload and it can consist of zero, one or more elements. In this case the payload consists of three elements C, D and E. To solve this problem of binding a formal parameter to several actual parameters we introduce the notion of a list. A list is an element which can contain zero or more elements. It is used where zero or more elements are needed to be bound to a formal parameter in a macro. Therefore the list $[C, D, E]$ is bound to M in the example. The list constructs used is removed after the expansion.

$$TLr(C, S, M, m) := (C, S, enc).\text{decrypt } enc \text{ as } \{M; m\}_K^{Serv} \text{ [orig } Cli] \text{ in } 0$$

Similarly, the TLr macro has four formal parameters, the first two C, S are bound as before. C is bound to M, it could have been bound to the empty list($[\,]$) or a list with more elements. M is the matching part of the decryption and m the variable binding part, therefore the list of variables $[x, y]$ is bound to m.

This example illustrate how the channel concept can be used to simplify LySa processes when modelling protocol using transport layers. A channel can be modelled by a macro which is instantiated when the process is expanded to remove the use of channels. Also the use of lists have been motivated and described. This simple example does, however, only cover some of the features needed to make this channel extension to LySAusable.

Additional features are required when dealing with more macros and macros which are instantiated several times.

Alpha Renaming In order to get the most precise analysis result a name or variable should only be used in one place or meaning in a model. The analysis is an over-approximation and hence will calculate the union of possible values of a variable. Because of this, it is beneficial to ensure that names and variables are not 'reused' with new semantic meaning in a process. A source of such a problem is when several instances of the same macro is used. Looking at the example above (Table 5.2 on page 64), if the *TL* macro was instantiated more than once, it would reuse the variable *enc*. To avoid this problem each channel is annotated with a string used as an identifier. This ID string is then on the actual expansion appended to each name and variable which is not a parameter or a restricted name (ν restriction).

Another problem is when processes are indexed (MetaLySa). Again looking at the example (Table 5.2 on page 64) if both the client and the server were indexed it would be necessary to apply these indices to the variable *enc* giving $enc_{i,j}$. When the indexed process is expanded each instance of *enc* would differ yielding a precise analysis result.

Global Names It is sometimes desirable to use names which are known globally and which do not need α-renaming. This could be a *dummy* name used only to enable variable bindings. There is no need to add id or indices to this name. To hinder the expansion from performing α-renaming a \$ can be prepended names. This means the value could be \$*dummy*.

Sequence Numbers When modelling transport layer protocols sequence numbers are often used to distinguish messages from the same session. Usually the first sequence number for a session is selected randomly and increased each time a message is sent/received. That is, each session will start with a unique sequence number and it is unlikely that the same sequence numbers should be used in concurrent sessions. As sequence numbers are used by transport layers they cannot be numbered before the expansion of channels. Instead sequence numbers are marked by prepending '#' to their name in the macro definitions. Then when the expansion has taken place they can be numbered. Also sequence numbers in a sending process (e.g. TLs) will have to match the corresponding sequence number in the matching receiving process (e.g. TLr) running in parallel.

Crypto-Points If encryptions or decryptions are annotated with crypto-points inside a macro they have to be α-renamed and numbered. As with sequence numbers crypto-points need to match in pairs of sending and receiving processes in parallel.

5.2 Extending LySa

We will now describe the extended syntax for CLySa. First of all a list construct has been added to terms, see Table 5.4 on the next page. The terms: name, variable and asymmetric keys deserve special attention. These types consist of a string describing the name and a list of indices. If the string is prefixed by '#' it is a sequence number. If the term is used inside a macro definition and the string is prefixed by '\$' it is treated as a publicly known global constant. This has the

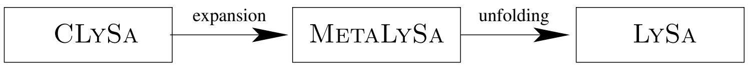


Figure 5.1: CLySa & MetaLySa & LySa

$E ::=$	*terms*
n	name ($n \in \mathcal{N}$)
x	variable ($x \in \mathcal{X}$)
m^+	
m^-	
$\{E_1, \cdots, E_k\}_{E_0}$	symetric encryption ($k \geq 0$)
$\{\|E_1, \cdots, E_k\|\}_{E_0}$	asymetric encryption ($k \geq 0$)
$[E_1, \cdots, E_k]$	List construct

Table 5.4: CLySa Terms

effect that the term is not subjected to alpha conversion. A new process type for macro instantiation has been added to processes (see Table 5.5). It requires that the macro m has been defined, meaning that it is in the environment $\mathcal{M}$. id is the string identifier added to the particular instance of the macro. $E_1, \cdots, E_k$ are the actual parameters for the macro.

The definitions of the macros themselves have their own syntax, see Table 5.6 on the next page. The name of the macro m and the formal arguments $arg_1, \cdots, arg_k$ are defined. The process P is what is inserted when a macro is instantiated. As any process ends with 0 when inserted this zero is where the next process element is placed. As explained before P should not contain any parallel constructs. To bind the macro definitions and the process we define a new top level type. A *program* both has the macro definitions and the actual process using the macros. A CLySa program first has the define keyword then some macro definitions then the in keyword and in the end the process. The program definition is shown in Table 5.7 on the following page.

$P ::=$	*processes* $\mathcal{P}$
0	nil
$\langle E_1, \cdots, E_k \rangle.P$	output
$(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k).P$	input (with matching)
$P_1 \| P_2$	parallel composition
$(\nu n)P$	restriction
$!P$	replication
decrypt E as $\{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}$ `in` P	symmetric decryption (with matching)
$(\nu_{\pm} n)P$	restriction
decrypt E as $\{\|E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\|\}_{E_0}$ `in` P	asymmetric decryption (with matching)
$m_{id}(E_1, \cdots, E_k).P$	Macro instance ($m \in \mathcal{M}$)

Table 5.5: CLySa Processes

$M ::=$	*macro definitions* $\mathcal{M}$
$m(arg_1, \cdots, arg_k) := P$	Macro definition

Table 5.6: CLySa macro definition

$G ::=$	*program*
define $M_1, \cdots, M_k$ in P	Macro definitions $\mathcal{M}$ used in $\mathcal{P}$

Table 5.7: CLySAprogram definition

5.3 Formalising the Expansion of CLySa

Up until now we have only described the expansion of CLySa to LySa informally. In this section we will formalise the conversion. This section will also introduce SML types to ease the reading of the implemented functions. During the expansion some environments are needed:

$\mathcal{M}$ a set of tuples with a macro name, its formal arguments and its process.

$\mathcal{B}$ a set of restricted names or formal arguments. Names in this environment should not be subjected to alpha conversion.

$\mathcal{I}$ a set of indices under which the current process is declared.

$\mathcal{C}$ a set of pairs of string and integer where the string is the name of a variable, name, etc.

To maintain readability and for the sake of briefness we have the semantics of the conversions are only stated for LySa without asymmetric encryptions and the extensions of MetaLySa. The full set of rules are stated in Appendix E on page 107. We formalise the expansion using inference rules.

5.3.1 Alpha Conversion

Terms are subjected to alpha conversion in order to generate unique names. The conversion is denoted as α shown in Table 5.8. Here the rules for asymmetric keys and variables have been left out, the full set of rules can be found in Appendix E.1 on page 107. A name is actually a tuple of a string as a name and a list of indices. The first rule, in Table 5.8, $NAME_{\notin}$ is used when the string name is not in the $\mathcal{B}$ environment. The id string is prepended to the string name and the index list is substituted with $\mathcal{I}$. If the string name is in $\mathcal{B}$ then the second rule is used and nothing is changed. The rule for encryption and for lists apply the conversion recursively. Crypto-points are also alpha converted and this conversion is denoted *cp* and shown in Table 5.9 on the next page. The conversion is similar to the one used for names, but no $\mathcal{B}$ environment is needed as all crypto-points should be converted. Additionally a "#" is prepended to the string name of each crypto-point to indicate that is should be numbered later, as described in Section 5.3.4 on page 72.

Table 5.8: Alpha conversion: α

$[NAME_{\notin}] \quad \mathcal{B}, \mathcal{I}, id \vdash (n, il) \xrightarrow{\alpha} (n\hat{\ }id, \mathcal{I})$

Continued on next page

Continued from previous page

If $n \notin \mathcal{B}$

$[NAME_{\in}]$ $\mathcal{B},\mathcal{I},id \vdash (n, il) \xrightarrow{\alpha} (n, il)$
If $n \in \mathcal{B}$

$$[ENC] \quad \frac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i \qquad \ell \xrightarrow{cp} \ell,\ \forall i \in\ [1;l]\ cp_i \xrightarrow{cp} cp'_i}{\mathcal{B},\mathcal{I},id \vdash \{E_1,\cdots,E_k\}^{\ell}_{E_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l] \xrightarrow{\alpha} \{E'_1,\cdots,E'_k\}^{\ell'}_{E'_0}\ [\mathsf{Dest}\, cp'_1,\cdots,cp'_l]}$$

$$[LIST] \quad \frac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i}{\mathcal{B},\mathcal{I},id \vdash [E_1,\cdots,E_k] \xrightarrow{\alpha} [E'_1,\cdots,E'_k]}$$

Table 5.9: Crypto-points

$$[CP] \quad \mathcal{I}, id \vdash (n, il) \xrightarrow{cp} (\#\hat{\ }n\hat{\ }id, \mathcal{I})$$

5.3.2 Substitution of Formal Parameters

When a macro is expanded the formal parameters are substituted with the actual parameters. The conversion uses an environment $\mathcal{A}$ to hold pairs of formal and actual parameters. For each name, variable or asymmetric key, if it exists in this environment it is substituted for the new value. When matching the formal parameters only the string part of the name is used. We use the operator $[\![x]\!]$ to denote the string name of x. Substitution is denoted s and is shown in Table 5.10.

Table 5.10: Substitution: s

$$[NAME] \quad \frac{([\![E]\!], E') \in \mathcal{A}}{\mathcal{A} \vdash E \xrightarrow{s} E'}$$

$$[ENC] \quad \frac{\forall i \in [0;k]\ \mathcal{A} \vdash E_i \xrightarrow{s} E'_i}{\mathcal{A} \vdash \{E_1,\cdots,E_k\}^{\ell}_{E_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l] \xrightarrow{s} \{E'_1,\cdots,E'_k\}^{\ell}_{E'_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l]}$$

$$[LIST] \quad \frac{\forall i \in [1;k]\ \mathcal{A} \vdash E_i \xrightarrow{s} E'_i}{\mathcal{A} \vdash [E_1,\cdots,E_k] \xrightarrow{s} [E'_1,\cdots,E'_k]}$$

5.3.3 Macros

Then expansion of macros are carried out in two steps. One where a macro is in the process of being inserted called *insertion*, denoted μ and shown in Table 5.11 on the following page. Another is where no macro is currently being inserted but any macro encountered is expanded called *expansion*, denoted η and shown in Table 5.12 on the next page. As described in Section 5.1 on page 64, in the scenarios we model it is not logical to use a parallel construct inside a macro definition. Macro insertion is, therefore not defined for parallel constructs.

Starting with the macro insertion a number of environments are defined. An environment with the defined macros $\mathcal{M}$, exceptions to alpha renaming $\mathcal{B}$, current indices $\mathcal{I}$, the formal and the actual parameters $\mathcal{A}$ and the identifier used with the current macro being inserted id. The asymmetric versions of *NEW* and *DEC* are treated as their symmetric counterparts. NIL, OUT, INP, NEW, DEC, BANG, PAR are all trivial in that they only apply alpha conversion and substitution to all terms and does a recursive call on the remaining process. One interesting thing with the rule for NEW is that is does not add the new name to $\mathcal{B}$. This is because it has itself already been subjected to α and s conversions, therefore subsequent occurrences should not be excluded. The rule for MACRO corresponds to a macro being used to define another macro. The challenge here is to expand the nested macro and then apply the current macro. First the name of the nested macro N is found in $\mathcal{M}$ hereby also giving the formal arguments $farg_1, \cdots, farg_k$. Then the new versions of the environments are calculated $\mathcal{B}'$ and $\mathcal{A}'$. First the nested macro insertion is applied to q giving q' and after this the current giving $q''.0$. The notation with $q''.0$ is used to show that the ending 0 is removed to allow p' to be prepended. It should be clear that a recursive macro definition must not appear since it would lead to an infinite loop of macro insertions and expansions.

Table 5.11: Macro insertion: μ

[*NIL*] $\quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash 0 \xrightarrow{\mu} 0$

[*OUT*]
$$\frac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E_i', \mathcal{A} \vdash E_i' \xrightarrow{s} E_i'', \quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash \langle E_1,\cdots,E_k\rangle.p \xrightarrow{\mu} \langle E_1',\cdots,E_k'\rangle.p'}$$

[*INP*]
$$\frac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E_i', \mathcal{A} \vdash E_i' \xrightarrow{s} E_i'' \quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash (E_1,\cdots;\cdots,E_k).p \xrightarrow{\mu} (E_1',\cdots;\cdots,E_k').p'}$$

[*NEW*]
$$\frac{\mathcal{B},\mathcal{I},id \vdash E \xrightarrow{\alpha} E', \mathcal{A} \vdash E' \xrightarrow{s} E'', \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash (\nu E).p \xrightarrow{\mu} (\nu E'').p'}$$

Continued on next page

Continued from previous page

$$[DEC]\quad \frac{\begin{array}{l}\mathcal{B},\mathcal{I},id \vdash E \xrightarrow{\alpha} E', \mathcal{A} \vdash E' \xrightarrow{s} E'',\\ \forall i \in [0;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E_i', \mathcal{A} \vdash E_i' \xrightarrow{s} E_i'',\\ \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p', \ell \xrightarrow{cp} \ell',\ \forall i \in [1;l]\ cp_i \xrightarrow{cp} cp_i'\end{array}}{\begin{array}{l}\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash \mathsf{decrypt}\, E\, \mathsf{as}\, \{E_1,\cdots;\cdots,E_k\}^{\ell}_{E_0}\\ [\mathsf{Orig}\, cp_1,\cdots,cp_l]\ \mathsf{in}\, p \xrightarrow{\mu}\\ \mathsf{decrypt}\, E'\, \mathsf{as}\, \{E_1',\cdots;\cdots,E_k'\}^{\ell'}_{E_0'}\\ [\mathsf{Orig}\, cp_1',\cdots,cp_l']\ \mathsf{in}\, p'\end{array}}$$

$$[MACRO]\quad \frac{\begin{array}{l}(N,(farg_1,\cdots,farg_k),q) \in \mathcal{M},\\ \forall arg \in (arg_1,\cdots,arg_k)\mathcal{B},\mathcal{I},id \vdash arg \xrightarrow{\alpha} arg', \mathcal{A} \vdash arg' \xrightarrow{s} arg'',\\ \mathcal{B}' := \mathcal{B} \cup \{farg_1,\cdots,farg_k\},\\ \mathcal{A}' := \{(farg_1,arg_1),\cdots,(farg_k,arg_k)\},\\ \mathcal{M},\mathcal{I},\mathcal{B}',\mathcal{A}',id' \vdash q \xrightarrow{\mu} q',\\ \mathcal{M},\mathcal{I},\mathcal{B},\mathcal{A},id \vdash q' \xrightarrow{\mu} q''.0,\ \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'\end{array}}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash N_{id'}(arg_1,\cdots,arg_k).p \xrightarrow{\mu} q''.p'}$$

Macro expansion is used to expand any macros in a process. When a macro is encountered in the process it will be expanded, and then the macro insertion will be used to do the actual insertion. Most of the rules for macro expansion are simple, in that they only apply the expansion to the rest of the process. One interesting thing is that in the rule for NEW the string name of the term is added to the $\mathcal{B}$ environment. The notation $\llbracket E \rrbracket$ is used to get the string name of a term. This of cause assumes the term to be a name or a variable. The rule for PAR_X (not shown) is the same as the one for PAR_I. Here the new index is added to the index environment $\mathcal{I}$. The rule for macros are, not surprisingly, the most complex. The name of the macro is found in the $\mathcal{M}$ environment and new $\mathcal{B}$ and $mathcalA$ environments are calculated. Then the macro insertion transformation is used to get the expanded process.

Table 5.12: Macro expansion: η

$$[NIL]\quad \mathcal{M},\mathcal{B},\mathcal{I} \vdash 0 \xrightarrow{\eta} 0$$

$$[OUT]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I} \vdash p \xrightarrow{\eta} p'}{\begin{array}{r}\mathcal{M},\mathcal{B},\mathcal{I} \vdash \langle E_1,\cdots,E_k\rangle.p \xrightarrow{\eta}\\ \langle E_1,\cdots,E_k\rangle.p'\end{array}}$$

$$[INP]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I} \vdash p \xrightarrow{\eta} p'}{\begin{array}{r}\mathcal{M},\mathcal{B},\mathcal{I} \vdash (E_1,\cdots;\cdots,E_k).p \xrightarrow{\eta}\\ (E_1,\cdots;\cdots,E_k).p'\end{array}}$$

$$[NEW]\quad \frac{\mathcal{M},\mathcal{B}\cup\{\llbracket E \rrbracket\},\mathcal{I} \vdash p \xrightarrow{\eta} p'}{\mathcal{M},\mathcal{B},\mathcal{I} \vdash (\nu E).p \xrightarrow{\mu} (\nu E).p'}$$

Continued on next page

Continued from previous page

$$[DEC]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I} \vdash p \xrightarrow{\eta} p'}{\begin{array}{r}\mathcal{M},\mathcal{B},\mathcal{I} \vdash \textsf{decrypt}\, E\, \textsf{as}\, \{E_1,\cdots;\cdots,E_k\}_{E_0}\ \textsf{in}\, p \xrightarrow{\eta} \\ \textsf{decrypt}\, E'\, \textsf{as}\, \{E_1,\cdots;\cdots,E_k\}_{E_0}\ \textsf{in}\, p'\end{array}}$$

$$[BANG]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I} \vdash p \xrightarrow{\eta} p'}{\mathcal{M},\mathcal{B},\mathcal{I} \vdash !p \xrightarrow{\eta} !p'}$$

$$[PAR]\quad \frac{\forall i \in [1;k]\ \mathcal{M},\mathcal{B},\mathcal{I} \vdash p_i \xrightarrow{\eta} p'_i}{\mathcal{M},\mathcal{B},\mathcal{I} \vdash p_1|\cdots|p_k \xrightarrow{\eta} p'_1|\cdots|p'_k}$$

$$[MACRO]\quad \frac{\begin{array}{l}(N,(farg_1,\cdots,farg_k),q) \in \mathcal{M},\\ \mathcal{B}' := \mathcal{B} \cup \{farg_1,\cdots,farg_k\},\\ \mathcal{A} := \{(farg_1,arg_1),\cdots,(farg_k,arg_k)\},\\ \mathcal{M},\mathcal{I},\mathcal{B}',\mathcal{A},id \vdash q \xrightarrow{\mu} q'.0,\ \mathcal{M},\mathcal{B},\mathcal{I} \vdash p \xrightarrow{\eta} p'\end{array}}{\mathcal{M},\mathcal{B},\mathcal{I} \vdash N_{id}(arg_1,\cdots,arg_k).p \xrightarrow{\eta} q'.p'}$$

5.3.4 Numbering

After macros have been expanded, names, variables and crypto-points which have a "#" as the first character in their string names have to be numbered. Meaning that the first occurrence should have the number 1 appended, the second 2, etc. As an example the process:

$$(\#num; inp;\ .)\langle \#num, out\rangle.0$$

has two instances of `#num`. After numbering this process would be:

$$(num1; inp;\ .)\langle num2, out\rangle.0$$

thereby removing "#" and appending numbers to the names. In order to allow numbered names to match between parallel processes, the numbering of names in each parallel process are independent. For example:

$$\langle \#seq, message\rangle.0|(\#seq; mess;\ .)0$$

here each "#seq" is in its own parallel process and the numbering of them will not affect each other.

$$\langle seq1, message\rangle.0|(seq1; mess;\ .)0$$

5.3.5 List Expansion

In CLySa we introduced a list construct. After the expansion of macros this construct has to be removed to allow the CLySa process to be converted to LySa. In LySa lists of terms are used many places, usually denoted $E_1,\cdots,E_k$. If a list construct occurs as one of these E_i elements it is inserted as the elements inside this list construct. That is if E_i is a list with the elements $E_{i,1}, E_{i,2}$ the list is expanded to $E_1, \cdots, E_{i,1}, E_{i,2}, \cdots, E_k$.

To formalise this we define two expansions. l which is a list expansion on a single term, and lt which is the expansion of a list of terms. The rule for lt is shown below. The full set of rules for lists, terms and processes are listed in Appendix E.6 on page 111.

$$\frac{\begin{array}{l}\forall i \in [1;k] \\ \quad \text{If } E_i = [E_{i,1}, \cdots, E_{i,j}] \\ \quad\ E_{i,1}, \cdots, E_{i,j} \xrightarrow{lt} E'_{i,1}, \cdots, E'_{i',j'} \\ \quad \text{Else } E_i \xrightarrow{l} E'_{i,1}\end{array}}{E_1, \cdots, E_k \xrightarrow{lt} E'_{1,1}, E'_{1,2}, \cdots, E'_{k,l}}$$

5.3.6 Variable Marking

Before macros are expanded it is unclear what names are indeed names and which are variables. After the macros are expanded the variables can be found by running through the process and picking any variable bindings up in an environment ($\mathcal{B}$). This is exactly the same functionality as the `bindVar`, `isVar` and `toVar` functions implemented in `lysaasciiio.sml` in the LySa-tool. The only difference is that now it is not done immediately after parsing the data but instead after macro expansion.

5.4 Implementation

The implementation of CLySa is build on top of LySa-tool [11]. The main focus of this project is not to implement a new version of LySa-tool but merely to provide the necessary features to allow us to model protocols using other protocols. From the outset the goal was to extend LySa-tool by touching as little as possible and in that sense CLySa is merely an extension much like MetaLySa is. The implementation is carried out by implementing new versions of: `lysa.sml`, `lysagrammar.grm`, `lysalexer.lex` and `lysaasciiio.sml` from LySa-tool. The new files have a “c” prepended. The lexer, grammar and asciiio files have not changed much as the only change needed was the addition of a few new data structures.

The environments used in the inference rules presented are here summarised and the types used are listed:

$\mathcal{M}$ a list of macro definitions: (string * string list * Proc) list

$\mathcal{B}$ a set of restricted names or formal arguments. string list

$\mathcal{I}$ a set of indices under which the current process is declared. string list

$\mathcal{C}$ a set of pairs of string names and integers: (string * int) list ref

5.4.1 Data Structure

The revised data structure for CLySA is shown below on Figure 5.2. Here `LIST` is added as a type of `Term` and `MACRO` is added as a type of `Proc`. Two new types are defined: `Macrodef` which is a definition of a macro and `Prog` which is a list of macro definitions and a process. The constructs listed under *experiment 1*, `LEAK` and `TEST` are not used in our implementation.

```
datatype Form = ORIG    of CP list
              | DEST    of CP list

datatype Term = NAME    of Name
              | NAMEP   of Name
              | NAMEM   of Name
              | VAR     of Var
              | ENC     of Term list * Term * CP * Form option
              | AENC    of Term list * Term * CP * Form option
              (**************************************************)
              (* Elements below are inserted by Jakob & Steffen   *)
              (**************************************************)
              | LIST    of Term list

datatype Proc = OUT     of Term list * Proc
              | INP     of Term list * Term list * Proc
              | DEC     of Term * Term list * Term list * Term *
                           CP * Form option * Proc
              | ADEC    of Term * Term list * Term list * Term *
                           CP * Form option * Proc
              | NEW     of Term * Proc
              | ANEW    of Term * Proc
              | BANG    of Proc
              | PAR     of Proc list
              | NIL

              (**************************************************)
              (* Elements below are conceptually part of Meta Lysa *)
              (**************************************************)

              | PAR_I   of string * int * Proc
              | PAR_X   of string * int * string * Proc
              | NEW_I   of ((string * int) list * Term) list * Proc
              | ANEW_I  of ((string * int) list * Term) list * Proc
              (**************************************************)
              (* Elements below are inserted for "Experiment 1"   *)
              (**************************************************)

              | LEAK    of Proc * (string * int) list
              | TEST    of string
              (**************************************************)
              (* Elements below are inserted by Jakob, Steffen    *)
              (**************************************************)
              | MACRO of string * string * Term list * Proc

type Macrodef = string * string list * Proc
type Prog = Macrodef list * Proc
```

Figure 5.2: Data Structure for CLySA

5.4.2 Combining the Expansions

In Section 5.3 on page 68, each part of the expansion of CLySA is explained. In the implementation the function `expandProg` combines them along with a function converting the CLySA data structure to the LySA data structure.

When using the program the CLySA-program is parsed from an ASCII file.

This results in a set of macros $\mathcal{M}$ and a CLYSA process P. Then the function `expandProg` is called with $\mathcal{M}$ and P as arguments. `expandProg` first uses macro expansion with $\mathcal{M}$, an empty $\mathcal{B}$ environment and an empty $\mathcal{I}$ environment. The resulting process is put through list expansion. After this the numbering expansion is used stating with an empty environment of numbers. Then variables are marked and finally the process is converted to the LYSA-process.

5.4.3 Tests

The implementation of the conversion form CLYSA to LYSA has been tested throughout the development. When a new macro was constructed it was necessary to verify that the macro was in fact expanding correctly. To supplement this and to provide concrete tests of unexpanded and expanded processes several tests have been performed, see Appendix G on page 127. Each test is mainly focused on one transformation. The tests have been designed to let them use all rules for a conversion at least once.

5.4.4 Correctness

CLYSA is as METALYSA is, an abbreviation of LYSA. CLYSA provides a convenient way to write LYSA processes. This means that we use the resulting LYSA process as an input to the analysis, and therefore it is this LYSA-process we actually analyse. The analysis result has no direct connection to METALYSA or CLYSA. This means that the semantics of CLYSA is defined in terms of LYSA and therefore the proves concerning the analysis of LYSA-processes still are valid for LYSA-processes obtained using either METALYSA or CLYSA (or both).

5.5 Example

We will now present a small example, designed to illustrate some of the features of CLYSA. Recall that a "$" in from of a name denotes a global constant. Most of the macros used in this report will use $\$dummy$ as a global constant. More specific it is used as a trick that allow us to bind values to variables (e.g. assignment see Section 4.5 on page 49. The trick is to use the `decrypt` term to bind a value to a variable. This often shortens complex processes and because it is used frequently a macro has been defined to simplify processes even more.

$$Bind(A, B) := \mathsf{decrypt}\ \{A\}_{\$dummy}[\mathsf{at}\,UCP]\ \mathsf{as}\ \{;\ B\}_{\$dummy}[\mathsf{at}\,UCP]\ \mathsf{in}\ \mathsf{0}$$

The `Bind` macro has two parameters A and B. The result of the macro is that the value of A is bound to the variable B.

The example we in Table 5.13 on the following page does perhaps not in it self hold any meaning, it should merely be seen as a demonstration of how macros are expanded.

In Table 5.13 we have defined three macros: Bind, Send and Rec. The Send macro use the Bind macro to do some variable binding. In the process first the name K is restricted and thereafter splitting into two parallel processes. Each

define $Bind(A, B) :=$ decrypt $\{A\}_{\$dummy}$[at UCP] as $\{;\ B\}_{\$dummy}$[at UCP] in
0
,
$Send(y) := Bind_{id}(y, Mess).$
$\langle\{Mess\}_K$[at C dest S]$\rangle.$
0
,
$Rec(x) := (;\ m).$
decrypt m as $\{;\ x\}_K$[at S orig C] in
0

in

$(\nu^n_{i=1\ldots n\, j=1\ldots n}\ K)$
$|^n_{i=1\ldots n}\ |^n_{j=1\ldots n\, j\neq i}\ (\nu\, M_{ij})$
$Send_a(M_{ij}).$
0
|
$|^n_{i=1\ldots n}\ |^n_{j=1\ldots n\, j\neq i}\ Rec_a(recieved_{ij}).$
0

Table 5.13: Example process before expansion

of these are indexed with i and j. The first use the Send macro and the second use the Rec macro.

The resulting process after the program has been expanded is showed in Table 5.14.

$(\nu^n_{i=1\ldots n\, j=1\ldots n}\ K)$
$|^n_{i=1\ldots n}\ |^n_{j=1\ldots n\, j\neq i}\ (\nu\, M_{ij})$
decrypt $\{M_{ij}\}_{\$dummy}$[at UCP] as $\{;\ Messa_{ij}\}_{\$dummy}$[at UCP] in
$\langle\{Messa_{ij}\}_K$[at $Ca1_{ij}$ dest $Sa1_{ij}$]$\rangle.$
0
|
$|^n_{i=1\ldots n}\ |^n_{j=1\ldots n\, j\neq i}\ (;\ ma_{ij}).$
decrypt ma_{ij} as $\{;\ recieved_{ij}\}_K$[at $Sa1_{ij}$ orig $Ca1_{ij}$] in
0

Table 5.14: Example process after expansion

5.6 TLS in CLySa

In Section 3.4.3 on page 32 extended protocol narrations for unilateral and bilateral TLS were developed. Now with the concept of channels in CLySa we can develop macros describing how to model the TLS transport layer in LySa. A protocol narration describes the sending and receiving parts of the protocol at the same time. In LySa however, the sending and receiving parts of each principal is described separately.

In Section 3.4.3 on page 32 we modelled the handshake and initialisation of the TLS protocol. To write this as macros we need to parameterise the protocol narration on the client C and the server S. As TLS uses a lot of messages to initialise a session we need different macros for initialising and subsequent messages. This all adds up to 10 different macros listed in Table 5.15.

TLS type	**Action**	**Role**	**Macro Name**
Unilateral	Initialisation	Client	TLSSInitC
Unilateral	Initialisation	Server	TLSSInitS
Bilateral	Initialisation	Client	TLSBInitC
Bilateral	Initialisation	Server	TLSBInitS
Both	Sending	Client	TLSSC
Both	Receiving	Client	TLSRC
Unilateral	Sending	Server	TLSSS
Unilateral	Receiving	Server	TLSRS
Bilateral	Sending	Server	TLSBSS
Bilateral	Sending	Server	TLSBRS

Table 5.15: Names of macros

All of these macros can be found in Appendix F.1 on page 115, but here we will only treat the `TLSBInitC` macro. This macro is used by a client to initialise a bilateral TLS connection. The messages in this macro are numbered with the same numbers used in the extended protocol narration for bilateral TLS Table 3.8 on page 34. The first line defines the macro and the formal arguments C and S. In the lines 1., 2.′ and 2.″ the client sends a nonce Nc to the server and receives a nonce yNs and a certificate $ycert$. Message 3. is implemented in many lines as both two variables are restricted and the macro *Bind* is used three times. *Bind* was introduced in the example in Section 5.5 on page 75. It is used to create meaningful names for values. In the last line 3. the client's certificate, the premaster secret, the certificate verify and the client finished messages are sent. The initialisation ends with the client receiving the server finished message. The implementation of this macro is shown in Table 5.16 on the following page.

5.7 SAML SSO in CLySa

Using the channels or macros defined for TLS we can now write the CLySa process for the SAML SSO protocol shown in Table 5.17 on page 79.

$TLSBInitC(C,S) := (\nu\, Nc)$

1. $\langle C, S, Nc\rangle.$
2.′ $(S, C;\ yNs, ycert).$
2.″ decrypt $ycert$ as $\{\!|S;\ yKS|\!\}_{CA^+}$[at UCP] in
3. $(\nu_\pm\ KC)$
3. $(\nu\, pm)$
3. $Bind_{id}(\{\!|Nc, yNs, pm|\!\}_{\$PRF}$[at UCP]$, yMaster).$
3. $Bind_{id}(\{\!|yMaster|\!\}_{\$PRF}$[at UCP]$, yMasterhash).$
3. $Bind_{id}(\{\!|yMaster|\!\}_{\$Key}$[at UCP]$, ySession).$
3. $\langle C, S, \{\!|C, KC^+|\!\}_{CA^-}$[at UCP]$, \{\!|\{\!|pm|\!\}_{yKS}$[at UCP]$|\!\}_{KC^-}$[at UCP],
   $\{\!|yMasterhash|\!\}_{KC^-}$[at UCP]$, \{\#Seq, yMasterhash\}_{ySession}$[at C dest S]$\rangle.$
4.′ $(S, C;\ ymhash).$
4.″ decrypt $ymhash$ as $\{\#Seq, yMasterhash;\ \}_{ySession}$[at C orig S] in
0

Table 5.16: The initialisation macro for bilateral TLS (Client-side)

$(\nu_{\pm}\ CA)\,($
$(\nu_{i=1}^{n}\ KU_i)$
$(\nu_{\pm}\ KLDY)\,($
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|S_0, KLDY^+|\!\}_{CA^-},$
$\quad \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
$)$
$|$
$|_{i=1}^{n}\ |_{j=1}^{n}\ |_{k=1}^{n}$
$!TLSSInitC_A(U_i, S_j).$
$TLSSC_A(U_i, S_j, D_k).$
$TLSRC_A(S_j, U_i, D_k, yArtifact_{ijk}).$
$TLSSInitC_B(U_i, D_k).$
$TLSSC_B(U_i, D_k, [S_j, yArtifact_{ijk}]).$
$(D_k, U_i;\ ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}$[at U_{ijk} orig D_{ijk}] in
0

$|$
$|_{i=0}^{n}\ |_{j=1}^{n}\ |_{k=0}^{n}$
$!TLSSInitS_A(U_i, S_j).$
$TLSRS_A(U_i, S_j, D_k, []).$
$(\nu\ Artifact)\,($
$TLSSS_A(S_j, U_i, [D_k, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_i]).$
0
$)$

$|$
$|_{i=0}^{n}\ |_{j=0}^{n}\ |_{k=1}^{n}$
$!TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_i).$
$(\nu\ Mess_{ijk})\,($
$\langle D_k, U_i, \{Mess_{ijk}\}_{zKU_i}$[at D_{ijk} dest U_{ijk}]$\rangle.$
0
$)$

$)$

Table 5.17: SAML SSO using TLS

Chapter 6

Analyses of Protocols

In this chapter we will present the results of our analyses on TLS and SAML SSO. In Section 6.1 we analyse our model of the TLS protocol. In Section 6.2 on page 83 we present the results from the analyses of different scenarios of the SAML SSO protocol.

6.1 Analysis of TLS

To analyse the TLS protocols we use a model with just two principals negotiating a connection and each sending a secret to each other.

Unilateral

For the unilateral version of TLS the ψ component of the analysis result is empty meaning that no errors were found. The LySa-process for this experiment can be found in Appendix F.2 on page 117. When applying the analysis without an attacker the variable bindings in ρ are the same as when an attacker is present. Since these bindings are the same in the two instances we can conclude the processes behave as expected. If this knowledge is combined with the fact that $\psi = \emptyset$ we also are able to conclude that the attacker cannot carry out any attacks affecting the confidentiality an authenticity properties of the protocol. To summarise this the LySa process behaves as intended.

$$\boxed{\begin{array}{lcl}
\psi & = & \emptyset \\
\rho(yCNEWSECRET_{i,j}) & = & CNEWSECRET_{i,j} \\
\rho(yCSECRET_{i,j}) & = & CSECRET_{i,j} \\
\rho(xSNEWSECRET_{i,j}) & = & SNEWSECRET_{i,j} \\
\rho(xSSECRET_{i,j}) & = & SSECRET_{i,j} \\
Secrets & \notin & \rho(z_\bullet)
\end{array}}$$

Bilateral

For the bilateral version of TLS (see Appendix F.3 on page 117) the analysis result contains errors. The ψ component is filled with pairs of the form $(l_\bullet, Sone\#_{i,j})$ and $(Sone\#_{i,j}, l_\bullet)$ where # is a number and i,j is indices. The errors stem from values encrypted in the server process and decrypted by the attacker or values encrypted by the attacker being decrypted by the server. This clearly show that something is wrong in server part of the protocol, which is, compared with the lack of errors in the unilateral version of TLS, not surprising, since the only changes in the process compared with unilateral TLS are in the server process. By using the ρ component of the result to inspect the variable bindings we are able to locate where the problems originate. The variable bindings: $\rho(xSSECRET_{i,j}) = SSECRET_{i,j}$ and $\rho(xSNEWSECRET_{i,j}) = SNEWSECRET$ show that the client will always get the correct values. These values are also known by the attacker. Since these messages are only sent on the network encrypted using the session key, the attacker must possess this key. The session key is constructed from three parts: the nonces N_c,N_s and pm the premaster secret. The nonces are send unencrypted on the network and therefore the attacker is able to change these nonces to whatever he wants them to be, resulting in that the variables holding these nonces could be bound to any value originating from the attacker: e.g. $V_\bullet \in \rho(yNs)$ for the client, and $V_\bullet \in \rho(xNc)$ for the server. This leaves the premaster secret pm, but it turns out that the variable holding the premaster secret at the server also could be bound to any value originating from the attacker: $V_\bullet \in \rho(xpm_{i,j})$. Now the attacker is able to determine the value of all the three elements the session key is created from, therefore the attacker is able to create a valid session key. The problem arises from the message 3 in TLS. In extended protocol narration this looked like:

$$\begin{array}{lll} 3. & C \rightarrow \quad & : C, S, \{|C, KC^+|\}_{CA^-}, \{|pm|\}_{yKS}, \{|\{|\{|N_c, y_{Ns}, pm|\}_{PRF}|\}_{PRF}|\}_{KC^-}, \\ & & \{Seq1, \{|\{|N_c, y_{Ns}, pm|\}_{PRF}|\}_{PRF}\}_{sessionkey}\,[\texttt{dest}\ \{S\}] \\ 3.' & \quad \rightarrow S & : x_c, x_s, x_{cert}, x_{cpm}, x_{certv}, x_{mh} \qquad [\texttt{check}\ x_s = S] \end{array}$$

In LySa it is:

$$\begin{array}{l} \ldots \\ \langle C, S, \{\!|C, KC^+|\!\}_{CA^-}[\mathsf{at}\,UCP], \\ \qquad \{\!|pm|\!\}_{yKS}[\mathsf{at}\,UCP], \\ \qquad \{\!|yMasterhash|\!\}_{KC^-}[\mathsf{at}\,UCP], \\ \qquad \{\#Seq, yMasterhash\}_{ySession}[\mathsf{at}\,C\ \mathsf{dest}\,S]\rangle. \\ \qquad \ldots \\ | \\ \ldots \\ (C, S;\ xcert, xcpm, xcertv, xmh). \\ \ldots \end{array}$$

Because the analysis is an independent attribute analysis (see Section 4.8.2 on page 59 for further details) it cannot relate the elements received. Furthermore `xcpm`, which is the premaster secret encrypted with the servers public key, is writable by anyone (since the public key is in fact "public"). This means that because the attacker knows the server's public key of, any value can be received as `xcpm`. The problem is that the analysis finds no connection between the ClientKeyExchange message and the CertificateVerify message. We claim that this reported error only occurs due to the independent attribute analysis, and that it cannot occur in a real implementation and therefore it is a *false positive*.

Helping the Analysis

One way to overcome the problem is to change the protocol slightly. A similar problem is noted in [9, pp. 126] where it is suggested to add the identity of the client to the `ClientKeyExchange` message thereby using the message: $\{|C, pm|\}_{KS^+}$. It turns out that this is not strong enough for our purposes as the analysis is still not be able to relate this message to the client's certificate. Therefore, we suggest signing the above message with the clients certificate giving the message $\{|\{|pm|\}_{KS^+}|\}_{KC^-}$. In the context from above this is:

$$\begin{array}{l}
\ldots \\
\langle C, S, \{\!|C, KC^+|\!\}_{CA^-}[\mathsf{at}\,UCP\,], \\
\qquad \{\!|\{\!|pm|\!\}_{yKS}[\mathsf{at}\,UCP\,]|\!\}_{KC^-}[\mathsf{at}\,UCP\,], \\
\qquad \{\!|yMasterhash|\!\}_{KC^-}[\mathsf{at}\,UCP\,], \\
\qquad \{\#Seq, yMasterhash\}_{ySession}[\mathsf{at}\,C\ \mathsf{dest}\,S\,]\rangle. \\
\qquad \ldots \\
| \\
\ldots \\
(C, S;\ xcert, xcpm, xcertv, xmh). \\
\ldots
\end{array}$$

This gives a stronger protocol; although, it does not prohibit anyone from reading the encrypted premaster secret. Actually, the remedy is along the lines of what is already done in the *CertificateVerify* message where the message is also signed by the client.

When analysing this new stronger version of the bilateral TLS protocol we discover no flaws. The result almost exactly the same as what we saw for the unilateral version of TLS. We will use this stronger version of the protocol in the experiments with SAML SSO. The macros forming this revised model of TLS is shown in Appendix F.1 on page 115.

6.2 SAML SSO

In Chapter 3 on page 19, we formally described the SAML SSO protocol and in Chapter 4 on page 37, we described the analysis tool LySa-tool. Using this tool we are now able to analyse the SAML SSO protocol.

Experiment 1

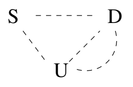


Figure 6.1: Experiment 1

The analysis of the SAML SSO protocol without any transport layer security, is depicted on Figure 6.1. The LySa process can be found in Appendix F.4 on page 118. It is obvious that this protocol does not provide any of the wanted security properties. None of the principals are authenticated and only the sixth message is encrypted. The key used for the sixth message could be eavesdropped when it was sent between the source and destination sites. One possible attack could be an attacker (A) acting as a user:

$$\begin{array}{lllll}
1. & A(U) & \rightarrow S & : D & \\
2. & S & \rightarrow A(U) & : Artifact & \\
3. & A(U) & \rightarrow D & : S, Artifact & \\
4. & D & \rightarrow S & : Artifact & \\
5. & S & \rightarrow D & : U, K_U & \\
5.' & & \rightarrow A & : U, K_U & : \text{Eavesdropped by A} \\
6. & D & \rightarrow A(U) & : \{Mess\}_{K_U} &
\end{array}$$

This is only one of the many possible attacks but as it is obvious this protocol is not secure at all we will not discuss this scenario further.

6.3 SAML SSO using TLS

The SAML specification [25] states three different types of security properties to be applied to the message transfers *authentication*, *message integrity* and *confidentiality*. It is also stated that when integrity or confidentiality is required SSL 3.0 [4] or TLS [13] must be used. For authentication either the unilateral or the bilateral version of TLS must be used.

The specification states that confidentiality and message integrity *must* be provided in step 1, 2 and 3 of the SAML SSO protocol. In the steps 4 and 5 confidentiality, integrity and bilateral authentication must be provided. No security properties are mandated for step 6.

Using the model of TLS it is possible extend the experiments with SAML SSO protocol using different combinations of the unilateral and bilateral versions of TLS between the three principals.

In order to model several instances of the protocol, running in parallel with one another, we will use indices as discussed in Section 4.6 on page 51. A process modelling a group of users is indexed with i, j, k meaning it is a user from the i'th user group talking to a source site from the j'th source site group and a destination site from the k'th destination site group. Some principals allow the attacker to interact with them as a normal principal in the protocol; this is modelled by starting the index at 0.

The scenario in which we model SAML SSO is shown in table 6.1 on the facing page. We allow the attacker to act as a user and as a destination site when communicating with a source site, and allowed him to act as a user when communicating with a destination site. That is, if a principal is accepting connections, which it has not initiated (as the source and destination sites do) the attacker can interact with these principals as a normal client. The user process is indexed with $i = 1, j = 1, k = 1$ reflecting that we assume that he only requests services from honest destination and source sites. The user is not allowed to contact an attacker directly as it would require the attacker to hold a trusted certificate. This gives a total of $n^3 + n \cdot (n+1)^2 + n^2 \cdot (n+1)$ parallel processes.

$|_{i=1}^{n}|_{j=1}^{n}|_{k=1}^{n}\ User_{i,j,k}$
$|$
$|_{i=0}^{n}|_{j=1}^{n}|_{k=0}^{n}\ Source\ site_{i,j,k}$
$|$
$|_{i=0}^{n}|_{j=1}^{n}|_{k=1}^{n}\ Destination\ site_{i,j,k}$

Table 6.1: The expansion of the analysis

The analysis itself is carried out for $n = 2$. This models two *groups* of users, destination sites and source sites where for each role we may think of one of the groups as being honest principals and the other group being malicious[1] principals. If possible one group will interfere with the other. As the different experiments analyses versions of the SAML SSO protocol which only differ slightly they have been illustrated using small drawings. In Table 6.3 on page 91 we summarise all the experiments depicted by drawings and a short description of the errors found, if any. The drawings show the three principals in SAML SSO: the User U, the Source site S and the Destination site D. The lines between the principals show how they exchange messages. If messages are sent without using any transport layer a dashed line is shown. Messages sent over the TLS transport layer is shown as a filled line. If a TLS connection is used either one or both of the principals are authenticated. This is shown with arrows pointing at authenticated principal(s).

Experiment 2

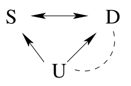


Figure 6.2: Experiment 2

The second experiment (depicted in figure 6.2) follows the recommendations from the SAML SSO documents closely. A bilateral TLS connection is used between the destination and the source sites. A unilateral TLS connection is used between the user and the source site and another between the user and destination site. This means the source and destination sites are authenticated to the user. The message in step 6 is however, not sent over a TLS connection, as no security properties are specified in the SAML documents [25, line 562–566]. The LySa process for this experiment can be found in Appendix F.5.

In this experiment the analysis finds several flaws. The ψ component contain the pairs $(D_{i,x,y}, U_{i,z,w})$, where $i, x, y, z, w \in [1, 2]$. This show that a message destined for U_i is decrypted at U_i but in the correct session. The attacker is able to switch the messages from step 6 between different sessions, because they are only encrypted with the users key and contain no information authenticating the sender.

[1]This group of malicious principals are able to communicate with the other group of principals because they all hold correct certificates.

6.	D	$\rightarrow A$:	$\{Mess\}_{K_U}$	: message in one session intercepted by an attacker
6.′	$A(D)$	$\rightarrow U$:	$\{Replay\}_{K_U}$	: replay of intercepted message in another session

This flaw is a result of our somewhat naive implementation of the message in step 6, but it shows that the specification should specify how messages in step 6 and later should be sent.

Experiment 3

$$\begin{tikzcd}
\mathrm{S} \arrow[rr, leftrightarrow] & & \mathrm{D} \\
& \mathrm{U} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 6.3: Experiment 3

The third experiment (depicted on figure 6.3) is an attempt to correct the flaw found in experiment 2. In this experiment the message in step 6 is sent over the unilateral TLS connection established in step 3. The setup of TLS connections are exactly the same as in experiment 2. The result of the analysis has an empty ψ component. $\rho(z_\bullet)$ does not contain $Mess$. At the same time in both the analysis result from using the analysis with an attack present and without him the same variable bindings occur for $yMess$. We conclude that this implementation of the protocol behaves as intended, and guarantees authenticity and confidentiality on the secret message. The LySa process can be found in Appendix F.6 on page 119.

ψ	$=$	$\emptyset$
$\rho(yMess_{i,j,k})$	$=$	$Mess_{i,j,k}$
$Mess_{i,j,k}$	$\notin$	$\rho(z_\bullet)$

Experiment 4

$$\begin{tikzcd}
\mathrm{S} & & \mathrm{D} \arrow[ll] \\
& \mathrm{U} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 6.4: Experiment 4

This experiment replaces the bilateral TLS connection between the destination and source sites by a unilateral TLS connection (depicted on figure 6.4). The analysis result now has a non-empty ψ component. $\psi = (D_{i,j,k}, l_\bullet)$, where $i, j, k \in 1, 2$. This means that the attacker can read the message sent by the destination site in step 6. The ρ component contains these values:

$\rho(z_\bullet)$	$=$	$KU_i, Mess_{i,j,k}, \ldots$
$\rho(xpmC_{i,j,k})$	$=$	$V_\bullet, \ldots$

This shows us that the attacker can learn KU_i for any session and as $V_\bullet$ is in $\rho(xpmC_{i,j,k})$ it can also decide the session key for the TLS connections between the source and destination sites using any value within the knowledge of the attacker. This is not surprising as the destination site is not authenticated to the source site. This means the following attack is possible:

$$\begin{array}{llll}
1. & A(U) \rightarrow S & : D \\
2. & S \rightarrow A(U) & : Artifact \\
4. & A(D) \rightarrow S & : Artifact \\
5. & S \rightarrow A(D) & : U, K_U
\end{array}$$

In step 1 the attacker impersonates the user and receives the artifact. The artifact is now used to impersonate D in step 4 and 5. This can be done because no certificate is required from D on the unilateral TLS connection. This compromises the key K_U as it is sent to the attacker. Using the compromised key the attacker is able to act as the user and at step 6 of the protocol decrypt all secret messages *Mess*. Note that the attacker is not able to impersonate the destination site D to the user U even though he possesses the key K_U, since authentication is required. The LySa-process describing this scenario can be found in Appendix F.7 on page 120

Experiment 5

In this experiment, depicted on Figure 6.5, we change the scenario a little. Now the key KU is generated by the source site in each session. This means the key is always fresh. This requires that the user is authenticated to the source site in the first step of the protocol. One easy way of doing this is to use a bilateral TLS connection between the user and the source site. This is particularly easy to model as we only have to change the TLS macro used.

The revised protocol narration is:

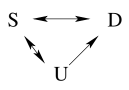


Figure 6.5: Experiment 5

The changes lie in messages 2. and 2.$'$ as the source site has to send the new key to the user.

ψ	$=$	$\emptyset$
$\rho(yMess_{i,j,k})$	$=$	$Mess_{i,j,k}$
$Mess_{i,j,k}$	$\notin$	$\rho(z_\bullet)$

The ψ component from the analysis result, using this protocol scenario as input, is empty. The values in ρ are present both when using the analysis with an attacker and without. This shows, that the protocol is correct in the sense that it can run without input from the attacker and that the attacker cannot change the variable binding. The LySa process can be found in Appendix F.8 on page 121.

1.	$U \rightarrow$	$: U, S, D$	
1.′	$\rightarrow S$	$: x_U, x_S, x_D$	[`check` $x_S = S$]
2.	$S \rightarrow$	$: S, x_U, x_D, Artifact, K_U$	
2.′	$\rightarrow U$	$: y_S, y_U, y_D, y_{Artifact}, xK_U$	[`check` $y_U = U$] [`check` $y_S = S,\ y_D = D$]
3.	$U \rightarrow$	$: U, y_D, S, y_{Artifact}$	
3.′	$\rightarrow D$	$: z_U, z_D, z_S, z_{Artifact}$	[`check` $z_D = D$]
4.	$D \rightarrow$	$: D, z_S, z_{Artifact}$	
4.′	$\rightarrow S$	$: x_D, x_S, x_{Artifact}$	[`check` $x_S = S$] [`check` $x_{Artifact} = Artifact$]
5.	$S \rightarrow$	$: S, D, K_U, U$	
5.′	$\rightarrow D$	$: z'_S, z_D, zK_U, z_{U'}$	[`check` $z_D = D$] [`check` $z_{U'} = z_U$] [`check` $z'_S = z_S$]
6.	$D \rightarrow$	$: D, U, \{Mess\}_{zK_U}$	[`dest` $\{U\}$]
6.′	$\rightarrow U$	$: y_D, y_U, yc_{Mess}$	[`check` $y_U = U$]
6.″	U	$:$ `decrypt` yc_{Mess} `as` $\{y_{Mess}\}_{xK_U}$	[`orig` $\{D\}$]

Table 6.2: Extended protocol narration for SAML SSO

Experiment 6

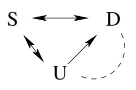


Figure 6.6: Experiment 6

Now as no errors were found in experiment 5 we change how the last (sixth) message is sent. In experiment 2 it was not sent over the TLS connection and this resulted in errors. We try the same in this scenario. This sixth experiment, depicted on Figure 6.6, only differs from experiment 5 in that message six is now not sent over a TLS connection (e.g. the message is only encrypted using the fresh key K_U).

ψ	$=$	$\emptyset$
$\rho(yMess_{i,j,k})$	$=$	$Mess_{i,j,k}$
$Mess_{i,j,k}$	$\notin$	$\rho(z_\bullet)$

Unlike experiment 2 with this experiment the analysis returns an empty ψ component and therefore finds no errors in the protocol. Also the value of ρ shown in the table above is the same running the analysis both with and without the attacker, as this result is exactly the same as fore the fifth experiment we conclude this protocol to be correct as well. The LySa process can be found in Appendix F.9 on page 122.

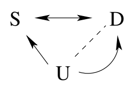


Figure 6.7: Experiment 7

Experiment 7

So far we have only looked at the SAML SSO source first[2] profile. SAML SSO defines another profile where it is the destination site with is contacted first. The only difference compared with the source first profile is that the protocol is initiated one step before. That is, if a user requests a service at a destination site, the user is redirected to a source site with the corresponding request. We extend our model by prepending a message from U to D and a redirect from D to U. The user process then uses the values of S and D provided in the redirect. We do not apply security properties on the redirect as it is not discussed in the SAML documents. This gives us this updated version of narration for SAML SSO:

$$
\begin{array}{lllll}
-2. & U \rightarrow & & : & \\
-2.' & & \rightarrow D & : & \\
-1. & D \rightarrow & & : & D, S \\
-1.' & & \rightarrow U & : & yD, yS \\
1. & U \rightarrow & & : & U, yS, yD \\
1.' & & \rightarrow S & : & x_U, x_S, x_D \quad [\texttt{check } x_S = S] \\
2. & S \rightarrow & & : & S, x_U, x_D, Artifact \\
2.' & & \rightarrow U & : & y_S, y_U, y_D, y_{Artifact} \quad [\texttt{check } y_U = U] \\
 & & & & \quad [\texttt{check } y_S = yS,\ y_D = yD] \\
3. & U \rightarrow & & : & U, y_D, yS, y_{Artifact} \\
3.' & & \rightarrow D & : & z_U, z_D, z_S, z_{Artifact} \quad [\texttt{check } z_D = D] \\
4. & D \rightarrow & & : & D, z_S, z_{Artifact} \\
4.' & & \rightarrow S & : & x_D, x_S, x_{Artifact} \quad [\texttt{check } x_S = S] \\
 & & & & \quad [\texttt{check } x_{Artifact} = Artifact] \\
5. & S \rightarrow & & : & S, D, K_U, U \\
5.' & & \rightarrow D & : & z'_S, z_D, z_{Key}, z_{U'} \quad [\texttt{check } z_D = D] \\
 & & & & \quad [\texttt{check } z_{U'} = z_U] \quad [\texttt{check } z'_S = z_S] \\
6. & D \rightarrow & & : & D, U, \{Mess\}_{z_{Key}}\, [\texttt{dest } \{U\}] \\
6.' & & \rightarrow U & : & y_D, y_U, yc_{Mess} \quad [\texttt{check } y_U = U] \\
6.'' & & U & : & \texttt{decrypt } yc_{Mess} \texttt{ as } \{y_{Mess}\}_{K_U}\, [\texttt{orig } \{D\}]
\end{array}
$$

The new messages −1 and −2 are not sent over a TLS connection. The rest of the messages are sent over TLS connections like experiment 3 where no errors were found. An unilateral TLS connection is used between U and S and another between U and D. A bilateral TLS connection is used between D and S. The LySa process for this narration can be found in Appendix F.10 on page 123.

The analysis of this process returns a non-empty ψ component in the result. Like the result in experiment 2, ψ contains pairs where only i match.

[2]Source-first means the User U contact the source site to be authenticate before contacting the destination site.

This means a user can get a message back from a wrong session. This is also seen by looking at the variable bindings. For example $\rho(yMess_{1,1,1}) = Mess_{1,1,1}, Mess_{1,1,2}, Mess_{1,2,1}, Mess_{1,2,2}$. The attacker though can not read $Mess$. The additional errors in ψ occur because the attacker can use his own certificate to play source site or destination site. If the attacker is not given a certificate these errors disappear.

$$\boxed{\begin{array}{lcl} \psi & = & (D_{i,j,k}, U_{i,l,m}) \cdots \\ \rho(yMess_{i,j,k}) & = & Mess_{i,l,m} \\ Mess_{i,j,k} & \notin & \rho(z_\bullet) \end{array}}$$

The errors are a result of the attacker changing the redirect message sent from D to U, see the narration below. The attacker could remove the redirect from D in message -1 and replace it with his own containing D', S' instead of D, S.

$$\begin{array}{llll} -2. & U & \rightarrow & : \\ -2.' & & \rightarrow D & : \\ -1. & D & \rightarrow & : D, S \\ -1.' & & \rightarrow A & : \ \text{This message is removed} \\ -1.* & A(D) & \rightarrow & : D', S' \\ -1.' & & \rightarrow U & : yD, yS \end{array}$$

Experiment 8

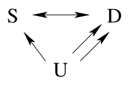


Figure 6.8: Experiment 8

To correct the errors found in experiment 7 the first messages (-1 and -2) should be sent over a TLS connection. This is a unilateral TLS connection between U and D. A connection like this already exist but we have chosen to use two different connections for communicating between U and D. The first one is used for the messages -1 and -2 and the other is used as usual for the rest of the messages. The LySa process for this experiment can be found in Appendix F.11 on page 124.

The analysis finds no errors in this scenario resulting in an empty ψ component. Still the variable binding of $yMess$ is $Mess$ both when using the analysis with and without the attacker.

$$\boxed{\begin{array}{lcl} \psi & = & \emptyset \\ \rho(yMess_{i,j,k}) & = & Mess_{i,j,k} \\ Mess_{i,j,k} & \notin & \rho(z_\bullet) \end{array}}$$

6.4 Summary of Experiments

The experiments are summarised in Table 6.3 on the facing page. Each of the experiments is either described as a scenario in the SAML specifications, or is

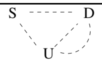
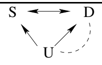
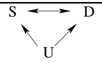
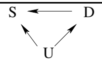
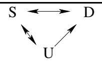
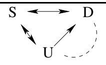
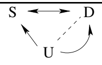
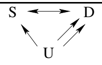
Ex #	Scenario	Description	$\rho(\mathbf{z}_\bullet)$	ψ
1		Multiple Errors	$Mess \in \rho(z_\bullet)$	$\psi \neq \emptyset$
2		User gets arbitrary result	$Mess \notin \rho(z_\bullet)$	$\psi \neq \emptyset$
3		No Errors	$Mess \notin \rho(z_\bullet)$	$\psi = \emptyset$
4		Compromised *Mess*	$Mess \in \rho(z_\bullet)$	$\psi \neq \emptyset$
5		No Errors	$Mess \notin \rho(z_\bullet)$	$\psi = \emptyset$
6		No Errors	$Mess \notin \rho(z_\bullet)$	$\psi = \emptyset$
7		User gets arbitrary result	$Mess \notin \rho(z_\bullet)$	$\psi \neq \emptyset$
8		No Errors	$Mess \notin \rho(z_\bullet)$	$\psi = \emptyset$

Table 6.3: Summary of Experiments

a scenario correcting a flaw found in another scenario.

Experiment 1 We analysed SAML SSO without any security and, as expected the analysis revealed several flaws.

Experiment 2 In experiment 2 we used the modelled of TLS to and analyse SAML SSO using the recommendations in the specification. If no security properties where applied to the message in step 6 of the protocol this protocol is also flawed.

Experiment 3 To correct the flaws from discovered in experiment 2, experiment 3 transmits the message in step 6 over the established TLS connection. The analysis finds no errors in this protocol.

Experiment 4 Experiment 4 replaces the bilateral TLS connection with an unilateral TLS connection only authenticating the source site. This protocol also has flaws thereby showing that the bilateral TLS connection is indeed required.

Experiment 5 The fifth experiment is of a slightly changed scenario where a fresh key is generated by the source site, and bilateral TLS connections are used not only between destination and source sites but also between users and source sites. As with experiment 3 we find no errors in this scenario.

Experiment 6 The sixth experiment is a combination of experiment 2 and experiment 5. In this experiment the last message is not sent over a TLS connection but unlike experiment 2 the analysis finds no errors because the key K_U is fresh in each session.

Experiment 7 Experiment 7 model the destination first profile of SAML SSO and finds several errors.

Experiment 8 The errors from experiment 7 are corrected in experiment 8 showing the importance of what security properties applied to redirects.

Chapter 7

Discussion

In this Chapter we will summarise our work and the results we have obtained on the analysis of different versions of the SAML Single Sign-On protocol. We will also briefly discuss how others have approached the problem of formal verification of web-protocols.

7.1 Scalability

The LySa-tool [8, 38] is used to analyse protocols described in the LySa process calculus. We have extended the LySa-calculus to be able to incorporate transport layers in the model of the protocols to be analysed. The protocols analysed in this thesis have all been modelled in the extended calculus CLySa. To be able to evaluate the scalability of the method used to analyse the SAML SSO protocol we here include some observations on the scalability of the LySa-tool.

During the course of this project we have at some points hit the limit of how complex processes the analysis could handle. It was clear almost from the beginning of this project that it would explore models of processes far larger than had been tried before. This inevitably lead to problems with scalability. To combat this Mikael Buchholtz constructed two optimisations to help us with the analysis.

- Firstly, the generation of ALFP logic was changed so that constraint generation of input and decryptions did not use nested implications.
- Secondly, a new way of generating labels was constructed. Before this optimisation each name or variable would get a new label for each occurrence of it in the process. This lead to a result which repeated itself as many times as the variable was used. Now each the same label is used for all occurrences of the same name or variable.

Each of these optimisations gave us a very large speedup of the time used by the analysis. Particularly the second optimisation had an enormous effect, alone cutting down the size of analysis result by up to twenty times.

Another problem encountered was the readability of the analysis result. We used the HTML output from the analysis, as it is generally nicely structured and easy to handle. The problem arose when the output got very large. To read the analysis result it was often necessary to jump from one label to another. To help this we have added links from label references to label definitions in the result allowing the result to be read faster.

Further, we had a problem with the result when many encryptions and decryptions were not annotated with crypto-points. This showed up in the result as a crypto-point with a destination or origin set containing all crypto-points. With the particular scenarios we used in this project this problem seem particularly visible as we use several TLS connections which each have many of these encryptions and decryptions without crypto-points. We defined that destination or origin sets that include $\ell_\bullet$, the crypto-point of the Dolev-Yao attacker, should not show up in the result. This gave us a ten fold decrease in the size of the result. We have not looked into how the analysis handles crypto-points but it is certainly not useful to have them in the result. Also it must cost time for the analysis to collect these crypto-points and it would therefore be preferred if the analysis did not collect them.

Scalability of LySa-tool

In this thesis the LySa processes analysed the upper limit of indices has been 2. This should be enough but it was certainly not possible to set this limit higher.

To see how LySa-tool scales with an increasing number of parallel processes we have constructed the graph shown in Figure 7.1 on the next page. As this thesis is mainly concerned with SAML SSO it is obviously interesting to see how it scales. In order to have something to compare this with, we have also included data points for the Wide Mouth Frog (WMF, see Appendix B.1 on page 101) protocol and the Needham-Schroeder symmetric key protocol (NSSK, see Appendix B.2 on page 102). The horizontal axis show the number of parallel processes analysed and the vertical axis show the number of seconds the analysis used. The vertical axis has a logarithmic scale. The data points from each of the three protocols are connected to allow us to see how the time increase as more parallel processes are modelled.

Clearly WMF and NSSK follow a polynomial trend. The last data point in each of these sets seem to jump up in the time used. This is properly caused by the New Jersey ML system almost running out of memory. The system can only allocate 1GB of memory and the last points gets close to this limit. As for SAML SSO it is clear that it is much more complex than the other two and the run time for it increases much faster than the other two. To take a closer look at SAML SSO we have constructed a graph where only data points from the lower end of the scale is used, see Figure 7.2 on the facing page. Here we can see that SAML SSO also seem to have a polynomial trend.

7.2 Evaluation

We have done a formal analysis of the SAML Single Sign-On protocol in different scenarios. We have found that the protocol provides authenticity and confidentiality, in the scenarios where a suitable usage of the transport layer protocol TLS is used. The specifications [24, 25, 26] are however somewhat unclear on how the TLS protocol should be applied, therefore we have studied different combination of this, resulting in different attacks.

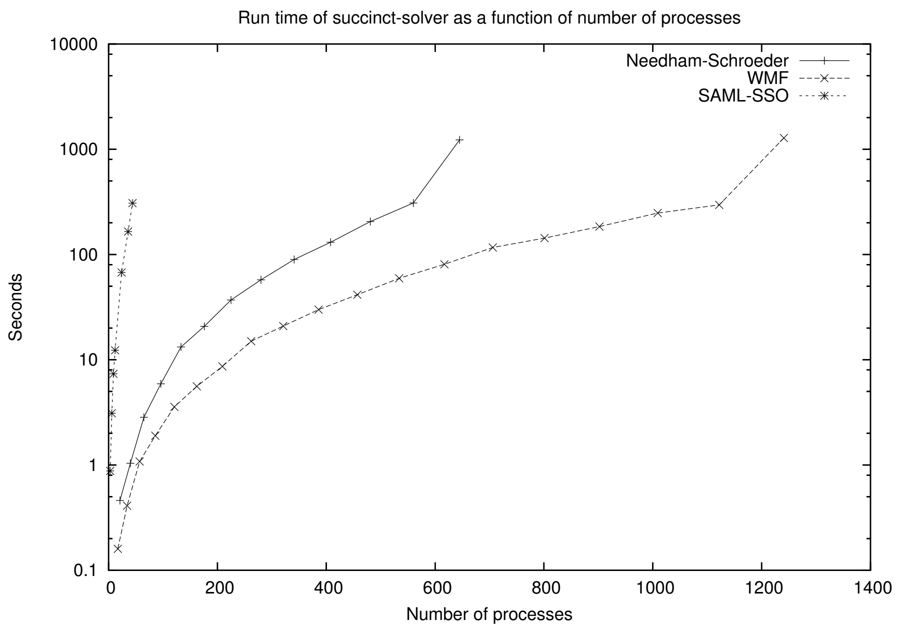


Figure 7.1: Running time of the analysis of the classic protocols

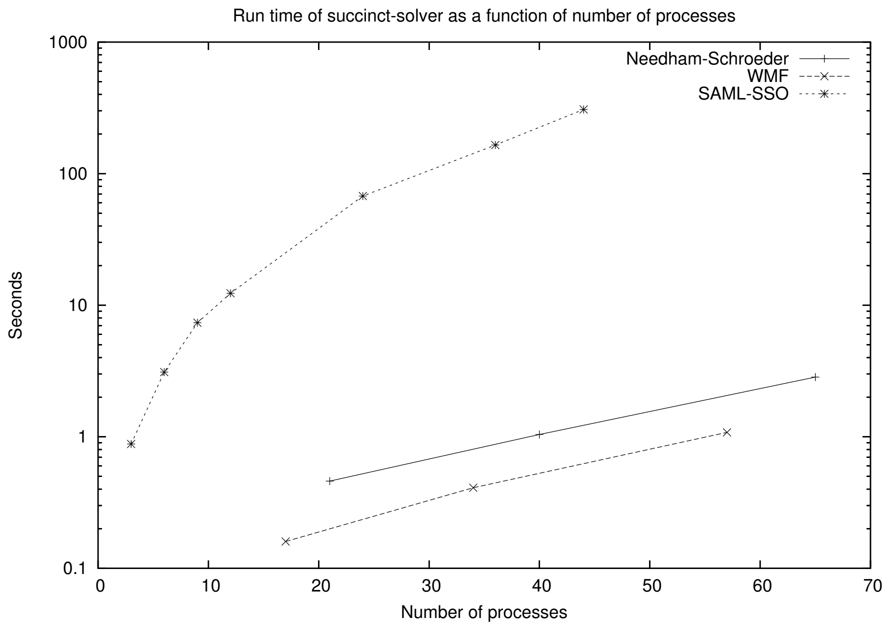


Figure 7.2: Running time of the analysis of the SAML SSO protocol

Specifications

Even though the attacks we discover in Section 6.2 on page 83 may seem obvious, they are not mentioned in the section discussing possible attacks on the SAML Single Sign-On protocols [26]. This fact emphasises the importance of a more thorough analysis of protocol specifications, since specifications are used as basis for the implementations of protocols, e.g. errors or terms which are misunderstood could lead to flawed implementations. This point is also stressed by Lawrence Paulson [34] in their analysis of the TLS protocol. In *Verifying policy-based security for web services* [7] the authors carry this thought even further by suggesting a XML-like language for specifying security goals of web-services. Using this language and the tool presented in [7] they are able to establish a direct link between the analysis result, the specification and the implementation of a protocol.

Even though the SAML Single Sign-On protocol is in general carefully designed, we believe it would have been even better, if the principles for design of cryptographic protocols presented by M. Arbadi and R. Needham in [2] were followed more closely.

Related work

In *Security Analysis of the SAML Single Sign-on Browser/Artifact Profile* [17] Thomas Groß presents his analysis of the SAML Single Sign-On. His analysis does not show the same attacks as we have found and presented in Section 6.2 on page 83. It seems that no formal methods or tools are used in the analysis of the protocol. He does through consider a broader spectrum of error than consider in this thesis. One attack scenario he claims exist which we do not consider is a HTTP referer attack. Security properties of different products are often compared using the Common Criteria [36]. Using these criteria we conclude that this work would be evaluated in the lower end of the *Evaluation Assurance Level* (EAL) scale. We conclude this on the basis that the work is structured, methodical and maybe even semi-formally described. In our thesis we use a formal method that is fully automated. We are convinced this would score a value on the EAL scale in the higher end.

Flexibility

All analysis result in this thesis are automatically generated. This flexibility is very useful, when small changes to the different scenarios are made, also our extension to LySa enabling us to use transport layer protocols has been very helpful.

At the outset of our thesis project, the available version of the specifications for the SAML SSO protocol was version 1.1. At some point later a draft version 2.0 was made available at the OASIS web-page [41]. To the best of our knowledge the only significant change in draft version 2.0 from version 1.1 of the specification is a new scenario. The addition of this scenario is that instead of using a longterm shared key known by the user and the server a new key is generated each time the user needs to be authenticated. Using the flexibility we where able to do analyses of combinations of this scenario, these are the scenario described as experiment 5 and 6 in Section 6.2 on page 83.

The specifications of the SAML SSO protocol focus on the scenario, where the user enters the source site first (*source-first*), if the destination site is contacted first (*destination-first*) the user should be transfered to the source site using a redirect. As this latter version could also be interesting for implementors of real web-services we analysed combinations of this version. Experiment 7 and 8 in Section 6.2 on page 83 show these results. Our analysis again revealed an attack not considered in the section discussing possible attacks on the SAML SSO protocol from [26]. As experiment 8 show this can be amended by using a TLS connection to transmit this redirect.

Further work

In the analysis of SAML SSO we use several instances of TLS connections. From the point of view of the analysis these connections are all the same. One obvious extension of our work would be to analyse one of these connections and get a result which is parametrised on the involved client and server. This would enable us to analyse a transport layer protocol once and the reuse this analysis result in every protocol using this transport layer protocol. Broadfoot and Lowe have in [10] suggested such a strategy.

This could also lead to a potential speedup of the LySa-tool allowing even larger protocols and scenarios to be analysed. This relates very much to the idea of verifying a library used in a programming language once. Subsequent analyses of programs using this library would then use this result in the analysis. This saves an analysis of all the functions used in the program.

Future work includes modelling even more aspects of realistic protocols as for example the Liberty Alliance protocols. With the growing size of not only the protocols but also the scenarios in which they are employed.

7.3 Conclusion

Using static analysis to validate protocols has already, with great success, been applied to many classical protocol scenarios [8]. In these scenarios there usually exist *one* server and two users who want to communicate in a secure manner. The work presented in this thesis shows that LySa and LySa-tool can be used to model and validate large protocols involving many principals in several different roles. The work carried out in this thesis is presented in the paper *Using static analysis to validate the SAML Single Sign-On protocol* [18] appearing on the Workshop on Issues in the Theory of Security (WITS'05) [43].

We have concentrated on the protocols defined by OASIS in the SAML documents. The importance of this line of work is evidenced by the comment from a SAML document [25]:

> *Before deployment, each combination of authentication, message integrity, and confidentiality mechanisms should be analysed for vulnerability in the context of the deployment environment.*

This leaves the programmer with a great burden, not least since history tells us that it is indeed very difficult to develop secure protocols. Without any tool support the best a programmer can do will typically be to follow the recommendations of the protocol designers.

We do pinpoint two errors or deficiencies in the specifications. As no recommendations on security properties for messages in the sixth step of the protocol is given, this can lead to a flawed interpretation of the protocol as seen in experiment 2. Also if the destination first profile of the SAML SSO protocol is used no recommendations are given regarding the first redirect. This can also lead to flaws as seen in experiment 7.

One can hardly blame the programmer who simply assume that if no security properties are recommended none are needed. As we have seen this results in a seriously flawed protocol. We are however able to correct these flaws by appropriate usage of the TLS protocol, and validate the authenticity, confidentiality and integrity properties.

Appendix A

ASCII input to LySa for WMF

```
/* Long term keys */
(new KA)(
(new KB)(

/* Initiators A*/
(
  (new K)
  <A, S, A, B, {K} : KA [at A1 dest {S1}]>.
  (new Secret)
  <A, B, {Secret} : K [at A2 dest {B2}]>.0 )
|


/* Server S */
(
  (A, S, A; xB, xMess).
  decrypt xMess as {;xKey} : KA [at S1 orig {A1}] in
  <S, xB, A, {xKey} : KB [at S2 dest {B1}]>. 0 )
|

/* Responder B*/
(
  (S, B; yA, yMess).
  decrypt yMess as {;yKey} : KB [at B1 orig {S2}] in
  (yA, B; yMessages).
  decrypt yMessages as {;ySecret} : yKey [at B2 orig {A2}] in 0)
))
```

Appendix B

Protocols used for benchmarking

B.1 WMF

```
/*******************************************************************/
/* Wide Mouthed Frog (without timestamp)                          */
/*                                                                 */
/* M. Burrows, M. Abadi, R. Needham: A logic of authentication,   */
/* ACM Transaction on Computer Systems, pp 18-36, 1990.           */
/*                                                                 */
/*                         Time-stamp: <17-03-2004 Mikael Buchholtz> */
/*******************************************************************/


/* Long term keys */
(new_{i=1} KL_{i})(


/* Initiators */
(
|_{i=1} |_{j=1\i}
! (new K_{i,j})
  <I_{i}, S, I_{i}, {I_{j}, K_{i,j}} : KL_{i} [at a1_{i,j} dest {s1_{i,j}}]>.
  (new mess_{i,j})
  <I_{i}, I_{j}, {mess_{i,j}} : K_{i,j} [at a2_{i,j} dest {b2_{i,j}}]>.0
)


|


/* Responder */
(
|_{j=1} |_{i=0}
! (S, I_{j}; y1_{i,j}).
  decrypt y1_{i,j} as {I_{i}; yk_{i,j}} : KL_{j} [at b1_{i,j} orig {s2_{i,j}}] in
  (I_{i},I_{j}; y2_{i,j}).
  decrypt y2_{i,j} as {;ym_{i,j}} : yk_{i,j} [at b2_{i,j} orig {a2_{i,j}}] in 0
)

|

/* Server */
(
|_{i=0} |_{j=0} !
  (I_{i}, S, I_{i}; z_{i,j}).
  decrypt z_{i,j} as {I_{j}; zk_{i,j}} : KL_{i} [at s1_{i,j} orig {a1_{i,j}}] in
  <S, I_{j}, {I_{i}, zk_{i,j}} : KL_{j} [at s2_{i,j} dest {b1_{i,j}}]>. 0 )
)
```

B.2 NSSK

```
/*****************************************************************/
/* Needham-Schroeder symmetric key                               */
/*                                                               */
/* R. Needham and M. Schroeder: Using encryption for authentication */
/* in large networks of computers. CACM, 21(12), pp. 993--999, 1978 */
/*                                                               */
/*                  Time-stamp: <17-03-2004 Mikael Buchholtz> */
/*****************************************************************/


/* Long term keys */
(new_{i=1} LK_{i})(


/* Initiators */
(
|_{i=1} |_{j=0} !(new NA_{i,j})
                 <I_{i}, S, I_{i}, S, NA_{i,j}>.
                 (S, I_{i}; x1_{i,j}).
                 decrypt x1_{i,j}
                      as {NA_{i,j},I_{j};x2_{i,j},x3_{i,j}} : LK_{i}
                      [at a1_{i,j} orig {s2_{i,j}}] in
                 <I_{i}, I_{j}, x3_{i,j}>.
                 (I_{j}, I_{i}; x4_{i,j}).
                 decrypt x4_{i,j}
                     as {;x5_{i,j}} : x2_{i,j}
                     [at a2_{i,j} orig {b2_{i,j}}] in
                 <I_{i}, I_{j}, { {x5_{i,j}}:succ } : x2_{i,j}
                                [at a3_{i,j} dest {b3_{i,j}}]>.0)

|

/* Responders */
(
|_{j=1} |_{i=0} !(I_{i}, I_{j}; y1_{i,j}).
                decrypt y1_{i,j}
                    as {I_{i}; y2_{i,j}} : LK_{j}
                    [at b1_{i,j} orig {s1_{i,j}}] in
                (new NB_{i,j})
                <I_{j}, I_{j}, {NB_{i,j}} : y2_{i,j}
                               [at b2_{i,j} dest { a2_{i,j} }]>.
                (I_{i}, I_{j}; y3_{i,j}).
                decrypt y3_{i,j}
                    as { {NB_{i,j}}:succ; } : y2_{i,j}
                    [at b3_{i,j} orig {a3_{i,j}}] in 0 )


|

/* Server */
(|_{i=0} |_{j=0} !(I_{i}, S, I_{i}, I_{j}; z1_{i,j}).
                 (new K_{i,j})
                 <S, I_{i},
                     { z1_{i,j}, I_{j}, K_{i,j},
                       {I_{i}, K_{i,j}} : LK_{j}
                       [at s1_{i,j} dest {b1_{i,j}}] } : LK_{i}
                     [at s2_{i,j} dest {a1_{i,j}}]>.0)

)
```

Appendix C

Users guide to CLySa

C.1 Installation Guide

To be able to carry out the analysis of the protocols as described in this thesis the software must be downloaded from the web-page:

- `http://www.student.dtu.dk/~s991471/clysa/`

Also the *New Jersey Standard ML* and the *Succinct Solver* must be downloaded and installed. These are also available at the web-page. Following line in the file `lysatool/sources.cm` must point to the installation of the Succinct Solver.

```
HORN/Formulas/sources.cm
```

In the directory `lysatool` the file `runhtml.sml` is used to specify which protocol to be analysed. In this file the line:

`val filename = "protocols/`*filename*";

In the subdirectory files for all protocols analysed in this theses are placed, the *filename* is used to indicate what protocol to be analysed. To run the analysis the interactive compiler of New Jersey Standard ML should be started using the command `sml`. If `lysatool` is the current directory the analysis is started by typing the following in the interactive compiler.

```
use ‘‘runhtml.sml’’;
```

When the analysis has finished the following files are produced:

Filename	Description
mlp.html	MetaLySa process for the parsed file
lp.html	LySa process for the parsed file
out.html	Analysis result for the parsed file

Appendix D

Overview of Source Files

analysis1.sml
clysa.sml
llysa.sml
lysa2llysa.sml
lysa.sml
mlysa2lysa.sml
run.sml
set.sml

Parsing CLySa from ASCII text

CLySa processes can be parsed from ASCII text using the grammar shown in Table 5.7 on page 68 . This grammar is an extension of what is presented in [11]. A new keyword $define$ has been added giving the following set of keywords:

```
as,at,orig,dest,define,decrypt,in,new,CPDY
```

Appendix E

From CLySa to LySa

Table E.1: Alpha α

$[NAME_{\notin}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n\hat{~}id,\mathcal{I})$ If $n \notin \mathcal{B}$
$[NAME_{\in}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n,il)$ If $n \in \mathcal{B}$
$[NAMEP_{\notin}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n\hat{~}id,\mathcal{I})$ If $n \notin \mathcal{B}$
$[NAMEP_{\in}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n,il)$ If $n \in \mathcal{B}$
$[NAMEM_{\notin}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n\hat{~}id,\mathcal{I})$ If $n \notin \mathcal{B}$
$[NAMEM_{\in}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n,il)$ If $n \in \mathcal{B}$
$[VAR_{\notin}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n\hat{~}id,\mathcal{I})$ If $n \notin \mathcal{B}$
$[VAR_{\in}]$	$\mathcal{B},\mathcal{I},id \vdash (n,il) \xrightarrow{\alpha} (n,il)$ If $n \in \mathcal{B}$
$[ENC]$	$\dfrac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i \quad \ell \xrightarrow{cp} \ell,\ \forall i \in [1;l]\ cp_i \xrightarrow{cp} cp'_i}{\mathcal{B},\mathcal{I},id \vdash \{E_1,\cdots,E_k\}^{\ell}_{E_0}\ [\mathsf{Dest}\,cp_1,\cdots,cp_l] \xrightarrow{\alpha} \{E'_1,\cdots,E'_k\}^{\ell'}_{E'_0}\ [\mathsf{Dest}\,cp'_1,\cdots,cp'_l]}$

Continued on next page

Continued from previous page

$$[AENC]\quad \frac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I}, id \vdash E_i \xrightarrow{\alpha} E_i' \qquad \ell \xrightarrow{cp} \ell,\ \forall i \in [1;l]\ cp_i \xrightarrow{cp} cp_i'}{\mathcal{B},\mathcal{I}, id \vdash \{|E_1,\cdots,E_k|\}^{\ell}_{E_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l] \xrightarrow{\alpha} \{|E_1',\cdots,E_k'|\}^{\ell'}_{E_0'}\ [\mathsf{Dest}\, cp_1',\cdots,cp_l']}$$

$$[LIST]\quad \frac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I}, id \vdash E_i \xrightarrow{\alpha} E_i'}{\mathcal{B},\mathcal{I}, id \vdash [E_1,\cdots,E_k] \xrightarrow{\alpha} [E_1',\cdots,E_k']}$$

Table E.2: Crypto-points

$$[CP]\quad \mathcal{I}, id \vdash (n, il) \xrightarrow{cp} (\#\hat{}n\hat{}id, \mathcal{I})$$

Table E.3: Substitution s

$$[NAME]\quad \frac{(\llbracket E \rrbracket, E') \in \mathcal{A}}{\mathcal{A} \vdash E \xrightarrow{s} E'}$$

$$[NAMEP]\quad \frac{(\llbracket E \rrbracket, E') \in \mathcal{A}}{\mathcal{A} \vdash E \xrightarrow{s} E'}$$

$$[NAMEM]\quad \frac{(\llbracket E \rrbracket, E') \in \mathcal{A}}{\mathcal{A} \vdash E \xrightarrow{s} E'}$$

$$[VAR]\quad \frac{(\llbracket E \rrbracket, E') \in \mathcal{A}}{\mathcal{A} \vdash E \xrightarrow{s} E'}$$

$$[ENC]\quad \frac{\forall i \in [0;k]\ \mathcal{A} \vdash E_i \xrightarrow{s} E_i'}{\mathcal{A} \vdash \{E_1,\cdots,E_k\}^{\ell}_{E_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l] \xrightarrow{s} \{E_1',\cdots,E_k'\}^{\ell}_{E_0'}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l]}$$

$$[AENC]\quad \frac{\forall i \in [0;k]\ \mathcal{A} \vdash E_i \xrightarrow{s} E_i'}{\mathcal{A} \vdash \{|E_1,\cdots,E_k|\}^{\ell}_{E_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l] \xrightarrow{s} \{|E_1',\cdots,E_k'|\}^{\ell}_{E_0'}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l]}$$

$$[LIST]\quad \frac{\forall i \in [1;k]\ \mathcal{A} \vdash E_i \xrightarrow{s} E_i'}{\mathcal{A} \vdash [E_1,\cdots,E_k] \xrightarrow{s} [E_1',\cdots,E_k']}$$

Table E.4: Macro insertion μ

$[NIL]$	$\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash 0 \xrightarrow{\mu} 0$
$[OUT]$	$\dfrac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i, \mathcal{A} \vdash E'_i \xrightarrow{s} E''_i,\quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash \langle E_1,\cdots,E_k\rangle.p \xrightarrow{\mu} \langle E'_1,\cdots,E'_k\rangle.p'}$
$[INP]$	$\dfrac{\forall i \in [1;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i, \mathcal{A} \vdash E'_i \xrightarrow{s} E''_i\quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash (E_1,\cdots;\cdots,E_k).p \xrightarrow{\mu} (E'_1,\cdots;\cdots,E'_k).p'}$
$[NEW]$	$\dfrac{\mathcal{B},\mathcal{I},id \vdash E \xrightarrow{\alpha} E', \mathcal{A} \vdash E' \xrightarrow{s} E'', \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash (\nu E).p \xrightarrow{\mu} (\nu E'').p'}$
$[ANEW]$	$\dfrac{\mathcal{B},\mathcal{I},id \vdash E \xrightarrow{\alpha} E', \mathcal{A} \vdash E' \xrightarrow{s} E'', \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p'}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash (\nu \pm E).p \xrightarrow{\mu} (\nu \pm E'').p'}$
$[DEC]$	$\dfrac{\mathcal{B},\mathcal{I},id \vdash E \xrightarrow{\alpha} E', \mathcal{A} \vdash E' \xrightarrow{s} E'',\quad \forall i \in [0;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i, \mathcal{A} \vdash E'_i \xrightarrow{s} E''_i,\quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p', \ell \xrightarrow{cp} \ell',\ \forall i \in [1;l]\ cp_i \xrightarrow{cp} cp'_i}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash \mathsf{decrypt}\, E\, \mathsf{as}\, \{E_1,\cdots;\cdots,E_k\}^{\ell}_{E_0}\ [\mathsf{Orig}\, cp_1,\cdots,cp_l]\ \mathsf{in}\, p \xrightarrow{\mu} \mathsf{decrypt}\, E'\, \mathsf{as}\, \{E'_1,\cdots;\cdots,E'_k\}^{\ell'}_{E'_0}\ [\mathsf{Orig}\, cp'_1,\cdots,cp'_l]\ \mathsf{in}\, p'}$
$[ADEC]$	$\dfrac{\mathcal{B},\mathcal{I},id \vdash E \xrightarrow{\alpha} E', \mathcal{A} \vdash E' \xrightarrow{s} E'',\quad \forall i \in [0;k]\ \mathcal{B},\mathcal{I},id \vdash E_i \xrightarrow{\alpha} E'_i, \mathcal{A} \vdash E'_i \xrightarrow{s} E''_i,\quad \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash p \xrightarrow{\mu} p', \ell \xrightarrow{cp} \ell',\ \forall i \in [1;l]\ cp_i \xrightarrow{cp} cp'_i}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id \vdash \mathsf{decrypt}\, E\, \mathsf{as}\, \{\lvert E_1,\cdots;\cdots,E_k\rvert\}^{\ell}_{E_0}\ [\mathsf{Orig}\, cp_1,\cdots,cp_l]\ \mathsf{in}\, p \xrightarrow{\mu} \mathsf{decrypt}\, E'\, \mathsf{as}\, \{\lvert E'_1,\cdots;\cdots,E'_k\rvert\}^{\ell'}_{E'_0}\ [\mathsf{Orig}\, cp'_1,\cdots,cp'_l]\ \mathsf{in}\, p'}$

Continued on next page

Continued from previous page

$$[MACRO]\quad \frac{\begin{array}{l}(N,(farg_1,\cdots,farg_k),q)\in\mathcal{M},\\ \forall arg\in(arg_1,\cdots,arg_k)\mathcal{B},\mathcal{I},id\vdash arg\xrightarrow{\alpha}arg',\mathcal{A}\vdash arg'\xrightarrow{s}arg'',\\ \mathcal{B}':=\mathcal{B}\cup\{farg_1,\cdots,farg_k\},\\ \mathcal{A}':=\{(farg_1,arg_1),\cdots,(farg_k,arg_k)\},\\ \mathcal{M},\mathcal{I},\mathcal{B}',\mathcal{A}',id'\vdash q\xrightarrow{\mu}q',\\ \mathcal{M},\mathcal{I},\mathcal{B},\mathcal{A},id\vdash q'\xrightarrow{\mu}q''.0,\ \mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id\vdash p\xrightarrow{\mu}p'\end{array}}{\mathcal{M},\mathcal{B},\mathcal{I},\mathcal{A},id\vdash N_{id'}(arg_1,\cdots,arg_k).p\xrightarrow{\mu}q''.p'}$$

Table E.5: Macro expansion η

$$[NIL]\quad \mathcal{M},\mathcal{B},\mathcal{I}\vdash 0\xrightarrow{\eta}0$$

$$[OUT]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash\langle E_1,\cdots,E_k\rangle.p\xrightarrow{\eta}\langle E_1,\cdots,E_k\rangle.p'}$$

$$[INP]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash(E_1,\cdots;\cdots,E_k).p\xrightarrow{\eta}(E_1,\cdots;\cdots,E_k).p'}$$

$$[NEW]\quad \frac{\mathcal{M},\mathcal{B}\cup\{\llbracket E\rrbracket\},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash(\nu E).p\xrightarrow{\mu}(\nu E).p'}$$

$$[ANEW]\quad \frac{\mathcal{M},\mathcal{B}\cup\{\llbracket E\rrbracket\},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash(\nu\pm E).p\xrightarrow{\mu}(\nu\pm E).p'}$$

$$[DEC]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash \textsf{decrypt}\,E\,\textsf{as}\,\{E_1,\cdots;\cdots,E_k\}_{E_0}\ \textsf{in}\,p\xrightarrow{\eta}\textsf{decrypt}\,E'\,\textsf{as}\,\{E_1,\cdots;\cdots,E_k\}_{E_0}\ \textsf{in}\,p'}$$

$$[ADEC]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash \textsf{decrypt}\,E\,\textsf{as}\,\{|E_1,\cdots;\cdots,E_k|\}_{E_0}\ \textsf{in}\,p\xrightarrow{\eta}\textsf{decrypt}\,E'\,\textsf{as}\,\{|E_1,\cdots;\cdots,E_k|\}_{E_0}\ \textsf{in}\,p'}$$

$$[BANG]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash !p\xrightarrow{\eta}!p'}$$

$$[PAR]\quad \frac{\forall i\in[1;k]\ \mathcal{M},\mathcal{B},\mathcal{I}\vdash p_i\xrightarrow{\eta}p_i'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash p_1|\cdots|p_k\xrightarrow{\eta}p_1'|\cdots|p_k'}$$

Continued on next page

Continued from previous page

$$[MACRO]\quad \frac{\begin{array}{l}(N,(farg_1,\cdots,farg_k),q)\in\mathcal{M},\\ \mathcal{B}':=\mathcal{B}\cup\{farg_1,\cdots,farg_k\},\\ \mathcal{A}:=\{(farg_1,arg_1),\cdots,(farg_k,arg_k)\},\\ \mathcal{M},\mathcal{I},\mathcal{B}',\mathcal{A},id\vdash q\xrightarrow{\mu}q'.0,\ \mathcal{M},\mathcal{B},\mathcal{I}\vdash p\xrightarrow{\eta}p'\end{array}}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash N_{id}(arg_1,\cdots,arg_k).p\xrightarrow{\eta}q'.p'}$$

$$[PAR\_I]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\cup\{i\}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash |_{i=n}\,p\xrightarrow{\eta}|_{i=n}\,p'}$$

$$[PAR\_X]\quad \frac{\mathcal{M},\mathcal{B},\mathcal{I}\cup\{i\}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash |_{i=a\backslash j}^{n}\,p\xrightarrow{\eta}|_{i=a\backslash j}^{n}\,p'}$$

$$[NEW\_I]\quad \frac{\mathcal{M},\mathcal{B}\cup\{E_i\},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash(\nu_{i=a}^{n}E_i).p\xrightarrow{\eta}(\nu_{i=a}^{n}E_i'').p'}$$

$$[ANEW\_I]\quad \frac{\mathcal{M},\mathcal{B}\cup\{E_i\},\mathcal{I}\vdash p\xrightarrow{\eta}p'}{\mathcal{M},\mathcal{B},\mathcal{I}\vdash(\nu_{i=a}^{n}\pm E_i).p\xrightarrow{\eta}(\nu_{i=a}^{n}\pm E_i'').p'}$$

Table E.6: List expansion on lists of terms lt

$$[TERM\ LISTS]\quad \frac{\begin{array}{l}\forall i\in[1;k]\\ \quad\text{If } E_i=[E_{i,1},\cdots,E_{i,j}]\\ \quad\quad E_{i,1},\cdots,E_{i,j}\xrightarrow{lt}E'_{i,1},\cdots,E'_{i',j'}\\ \quad\text{Else } E_i\xrightarrow{l}E'_{i,1}\end{array}}{E_1,\cdots,E_k\xrightarrow{lt}E'_{1,1},E'_{1,2},\cdots,E'_{k,l}}$$

Table E.7: List expansion application on terms l

$$[NAME]\quad E\xrightarrow{l}E$$

$$[NAMEP]\quad E\xrightarrow{l}E$$

$$[NAMEM]\quad E\xrightarrow{l}E$$

$$[VAR]\quad E\xrightarrow{l}E$$

$$[ENC]\quad \frac{E_1,\cdots,E_k\xrightarrow{lt}E'_1,\cdots,E'_j,\quad E_0\xrightarrow{l}E'_0}{\begin{array}{l}\mathcal{A}\vdash\{E_1,\cdots,E_k\}_{E_0}^{\ell}\ [\mathsf{Dest}\,cp_1,\cdots,cp_l]\xrightarrow{s}\\ \quad\{E'_1,\cdots,E'_j\}_{E'_0}^{\ell}\ [\mathsf{Dest}\,cp_1,\cdots,cp_l]\end{array}}$$

Continued on next page

Continued from previous page

$$[AENC]\quad \frac{E_1,\cdots,E_k \xrightarrow{lt} E'_1,\cdots,E'_j,\quad E_0 \xrightarrow{l} E'_0}{\mathcal{A}\vdash\{|E_1,\cdots,E_k|\}^{\ell}_{E_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l] \xrightarrow{s} \{|E'_1,\cdots,E'_j|\}^{\ell}_{E'_0}\ [\mathsf{Dest}\, cp_1,\cdots,cp_l]}$$

Table E.8: List expansion application on processes lp

$$[NIL]\quad 0 \xrightarrow{lp} 0$$

$$[OUT]\quad \frac{E_1,\cdots,E_k \xrightarrow{lt} E'_1,\cdots,E'_j,\quad p \xrightarrow{lp} p'}{\langle E_1,\cdots,E_k\rangle.p \xrightarrow{\eta} \langle E'_1,\cdots,E'_j\rangle.p'}$$

$$[INP]\quad \frac{E_1,\cdots,E_k \xrightarrow{lt} E'_1,\cdots,E'_j,\quad p \xrightarrow{lp} p'}{(E_1,\cdots;\cdots,E_k).p \xrightarrow{lp} (E'_1,\cdots;\cdots,E'_j).p'}$$

$$[NEW]\quad \frac{p \xrightarrow{lp} p'}{(\nu E).p \xrightarrow{lp} (\nu E).p'}$$

$$[ANEW]\quad \frac{p \xrightarrow{lp} p'}{(\nu \pm E).p \xrightarrow{lp} (\nu \pm E).p'}$$

$$[DEC]\quad \frac{E \xrightarrow{l} E', E_1,\cdots,E_k \xrightarrow{lt} E'_1,\cdots,E'_j, E_0 \xrightarrow{l} E'_0\quad p \xrightarrow{lp} p'}{\mathsf{decrypt}\, E\ \mathsf{as}\,\{E_1,\cdots;\cdots,E_k\}_{E_0}\ \mathsf{in}\, p \xrightarrow{lp} \mathsf{decrypt}\, E'\ \mathsf{as}\,\{E'_1,\cdots;\cdots,E'_j\}_{E'_0}\ \mathsf{in}\, p'}$$

$$[ADEC]\quad \frac{E \xrightarrow{l} E', E_1,\cdots,E_k \xrightarrow{lt} E'_1,\cdots,E'_j, E_0 \xrightarrow{l} E'_0\quad p \xrightarrow{lp} p'}{\mathsf{decrypt}\, E\ \mathsf{as}\,\{|E_1,\cdots;\cdots,E_k|\}_{E_0}\ \mathsf{in}\, p \xrightarrow{lp} \mathsf{decrypt}\, E'\ \mathsf{as}\,\{|E'_1,\cdots;\cdots,E'_j|\}_{E'_0}\ \mathsf{in}\, p'}$$

$$[BANG]\quad \frac{p \xrightarrow{lp} p'}{!p \xrightarrow{lp} !p'}$$

$$[PAR]\quad \frac{\forall i \in [1;k]\ p_i \xrightarrow{lp} p'_i}{p_1|\cdots|p_k \xrightarrow{lp} p'_1|\cdots|p'_k}$$

$$[PAR\_I]\quad \frac{p \xrightarrow{lp} p'}{|_{i=n}\, p \xrightarrow{lp} |_{i=n}\, p'}$$

Continued on next page

Continued from previous page

$$[PAR\_X] \quad \frac{p \xrightarrow{lp} p'}{|_{i=a\backslash j}^{n}\, p \xrightarrow{lp} |_{i=a\backslash j}^{n}\, p'}$$

$$[NEW\_I] \quad \frac{p \xrightarrow{lp} p'}{(\nu_{i=a}^{n} E_i).p \xrightarrow{lp} (\nu_{i=a}^{n} E_i'').p'}$$

$$[ANEW\_I] \quad \frac{p \xrightarrow{lp} p'}{(\nu_{i=a}^{n} \pm E_i).p \xrightarrow{lp} (\nu_{i=a}^{n} \pm E_i'').p'}$$

Appendix F

Protocols in LySa

F.1 TLS macros

define $Bind(A,B)$:= decrypt $\{A\}_{\$dummy}[\text{at}\, UCP]$ as $\{;\, B\}_{\$dummy}[\text{at}\, UCP]$ in
0
,
$TLSSInitC(C,S) := (\nu\, Nc)$
$\langle C, S, Nc\rangle.$
$(S, C;\ yNs, ycert).$
decrypt $ycert$ as $\{\!|S;\ yKS|\!\}_{CA^+}[\text{at}\, UCP]$ in
$(\nu\, pm)$
$Bind_{id}(\{\!|Nc, yNs, pm|\!\}_{\$PRF}[\text{at}\, UCP], yMaster).$
$Bind_{id}(\{\!|yMaster|\!\}_{\$PRF}[\text{at}\, UCP], yMasterhash).$
$Bind_{id}(\{\!|yMaster|\!\}_{\$Key}[\text{at}\, UCP], ySession).$
$\langle C, S, \{\!|pm|\!\}_{yKS}[\text{at}\, UCP], \{\#Seq, yMasterhash\}_{ySession}[\text{at}\, C\ \text{dest}\, S]\rangle.$
$(S, C;\ ymhash).$
decrypt $ymhash$ as $\{\#Seq, yMasterhash;\ \}_{ySession}[\text{at}\, C\ \text{orig}\, S]$ in
0
,
$TLSBInitC(C,S) := (\nu\, Nc)$
$\langle C, S, Nc\rangle.$
$(S, C;\ yNs, ycert).$
decrypt $ycert$ as $\{\!|S;\ yKS|\!\}_{CA^+}[\text{at}\, UCP]$ in
$(\nu_\pm\ KC)$
$(\nu\, pm)$
$Bind_{id}(\{\!|Nc, yNs, pm|\!\}_{\$PRF}[\text{at}\, UCP], yMaster).$
$Bind_{id}(\{\!|yMaster|\!\}_{\$PRF}[\text{at}\, UCP], yMasterhash).$
$Bind_{id}(\{\!|yMaster|\!\}_{\$Key}[\text{at}\, UCP], ySession).$
$\langle C, S, \{\!|C, KC^+|\!\}_{CA^-}[\text{at}\, UCP], \{\!|\{\!|pm|\!\}_{yKS}[\text{at}\, UCP]|\!\}_{KC^-}[\text{at}\, UCP],$
$\quad \{\!|yMasterhash|\!\}_{KC^-}[\text{at}\, UCP], \{\#Seq, yMasterhash\}_{ySession}[\text{at}\, C\ \text{dest}\, S]\rangle.$
$(S, C;\ ymhash).$
decrypt $ymhash$ as $\{\#Seq, yMasterhash;\ \}_{ySession}[\text{at}\, C\ \text{orig}\, S]$ in
0
,
$TLSSC(C, S, Mess) := \langle C, S, \{\#Seq, Mess\}_{ySession}[\text{at}\, C\ \text{dest}\, S]\rangle.$

0
,
$TLSRC(S, C, MMess, Mess) := (C, S;\ ycmess).$
decrypt $ycmess$ as $\{\#Seq, MMess;\ Mess\}_{ySession}[\mathsf{at}\ C\ \mathsf{orig}\ S]$ in
0
,
$TLSSInitS(C, S) := (C, S;\ xNc).$
$(\nu_{\pm}\ KS)$
$(\nu\ Ns)$
$\langle S, C, Ns, \{|S, KS^{+}|\}_{CA^{-}}[\mathsf{at}\ UCP]\rangle.$
$(C, S;\ xcpm, xmh).$
decrypt $xcpm$ as $\{|;\ xpm|\}_{KS^{-}}[\mathsf{at}\ UCP]$ in
$Bind_{id}(\{|xNc, Ns, xpm|\}_{\$PRF}[\mathsf{at}\ UCP], xMaster).$
$Bind_{id}(\{|xMaster|\}_{\$PRF}[\mathsf{at}\ UCP], xMasterhash).$
$Bind_{id}(\{|xMaster|\}_{\$Key}[\mathsf{at}\ UCP], xSession).$
decrypt xmh as $\{\#Seq, xMasterhash;\ \}_{xSession}[\mathsf{at}\ S]$ in
$\langle S, C, \{\#Seq, xMasterhash\}_{xSession}[\mathsf{at}\ S]\rangle.$
0
,
$TLSBInitS(C, S) := (C, S;\ xNc).$
$(\nu_{\pm}\ KS)$
$(\nu\ Ns)$
$\langle S, C, Ns, \{|S, KS^{+}|\}_{CA^{-}}[\mathsf{at}\ UCP]\rangle.$
$(C, S;\ xcert, xccpm, xcertv, xmh).$
decrypt $xcert$ as $\{|C;\ xKC|\}_{CA^{+}}[\mathsf{at}\ UCP]$ in
decrypt $xccpm$ as $\{|;\ xcpm|\}_{xKC}[\mathsf{at}\ UCP]$ in
decrypt $xcpm$ as $\{|;\ xpm|\}_{KS^{-}}[\mathsf{at}\ UCP]$ in
$Bind_{id}(\{|xNc, Ns, xpm|\}_{\$PRF}[\mathsf{at}\ UCP], xMaster).$
$Bind_{id}(\{|xMaster|\}_{\$PRF}[\mathsf{at}\ UCP], xMasterhash).$
decrypt $xcertv$ as $\{|xMasterhash;\ |\}_{xKC}[\mathsf{at}\ UCP]$ in
$Bind_{id}(\{|xMaster|\}_{\$Key}[\mathsf{at}\ UCP], xSession).$
decrypt xmh as $\{\#Seq, xMasterhash;\ \}_{xSession}[\mathsf{at}\ S\ \mathsf{orig}\ C]$ in
$\langle S, C, \{\#Seq, xMasterhash\}_{xSession}[\mathsf{at}\ S\ \mathsf{dest}\ C]\rangle.$
0
,
$TLSSS(S, C, Mess) := \langle S, C, \{\#Seq, Mess\}_{xSession}[\mathsf{at}\ S]\rangle.$
0
,
$TLSRS(C, S, MMess, Mess) := (C, S;\ xcmess).$
decrypt $xcmess$ as $\{\#Seq, MMess;\ Mess\}_{xSession}[\mathsf{at}\ S]$ in
0
,
$TLSBSS(S, C, Mess) := \langle S, C, \{\#Seq, Mess\}_{xSession}[\mathsf{at}\ S\ \mathsf{dest}\ C]\rangle.$
0
,
$TLSBRS(C, S, MMess, Mess) := (C, S;\ xcmess).$
decrypt $xcmess$ as $\{\#Seq, MMess;\ Mess\}_{xSession}[\mathsf{at}\ S\ \mathsf{orig}\ C]$ in
0

F.2 test_tls

$(\nu_{\pm}\ \mathit{CA})\,($
$(\nu_{\pm}\ \mathit{KLDY})\,($
$\langle CA^{+}, KLDY^{-}, \{\!|LDY, KLDY^{+}|\!\}_{CA^{-}}, \{\!|U_0, KLDY^{+}|\!\}_{CA^{-}},$
$\quad \{\!|S_0, KLDY^{+}|\!\}_{CA^{-}}, \{\!|D_0, KLDY^{+}|\!\}_{CA^{-}}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n}\,|_{j=0}^{n}\,!TLSSInitC_{one}(A_i, B_j).$
$(\nu\ \mathit{CSECRET}_{ij})$
$TLSSC_{one}(A_i, B_j, \mathit{CSECRET}_{ij}).$
$TLSRC_{one}(A_i, B_j, [], \mathit{xSSECRET}_{ij}).$
$(\nu\ \mathit{CNEWSECRET}_{ij})$
$TLSSC_{one}(A_i, B_j, \mathit{CNEWSECRET}_{ij}).$
$TLSRC_{one}(A_i, B_j, [], \mathit{xSNEWSECRET}_{ij}).$
0

|
$|_{i=0}^{n}\,|_{j=1}^{n}\,!TLSSInitS_{one}(A_i, B_j).$
$TLSRS_{one}(A_i, B_j, [], \mathit{yCSECRET}_{ij}).$
$(\nu\ \mathit{SSECRET}_{ij})$
$TLSSS_{one}(A_i, B_j, \mathit{SSECRET}_{ij}).$
$TLSRS_{one}(A_i, B_j, [], \mathit{yCNEWSECRET}_{ij}).$
$(\nu\ \mathit{SNEWSECRET}_{ij})$
$TLSSS_{one}(A_i, B_j, \mathit{SNEWSECRET}_{ij}).$
0
)

F.3 test_tls_bilateral

$(\nu_{\pm}\ \mathit{CA})\,($
$(\nu_{\pm}\ \mathit{KLDY})\,($
$\langle CA^{+}, KLDY^{-}, \{\!|LDY, KLDY^{+}|\!\}_{CA^{-}}, \{\!|U_0, KLDY^{+}|\!\}_{CA^{-}},$
$\quad \{\!|S_0, KLDY^{+}|\!\}_{CA^{-}}, \{\!|D_0, KLDY^{+}|\!\}_{CA^{-}}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n}\,|_{j=0}^{n}\,!TLSBInitC_{one}(A_i, B_j).$
$(\nu\ \mathit{CSECRET}_{ij})$
$TLSSC_{one}(A_i, B_j, \mathit{CSECRET}_{ij}).$
$TLSRC_{one}(A_i, B_j, [], \mathit{xSSECRET}_{ij}).$
$(\nu\ \mathit{CNEWSECRET}_{ij})$
$TLSSC_{one}(A_i, B_j, \mathit{CNEWSECRET}_{ij}).$
$TLSRC_{one}(A_i, B_j, [], \mathit{xSNEWSECRET}_{ij}).$
0

|
$|_{i=0}^{n}\,|_{j=1}^{n}\,!TLSBInitS_{one}(A_i, B_j).$

$TLSBRS_{one}(A_i, B_j, [], yCSECRET_{ij})$.
$(\nu\, SSECRET_{ij})$
$TLSBSS_{one}(A_i, B_j, SSECRET_{ij})$.
$TLSBRS_{one}(A_i, B_j, [], yCNEWSECRET_{ij})$.
$(\nu\, SNEWSECRET_{ij})$
$TLSBSS_{one}(A_i, B_j, SNEWSECRET_{ij})$.
0
)

F.4 LySa process for experiment 1

$(\nu_{\pm}\ CA)$ (
$(\nu_{i=1}^{n}\ KU_i)$
$(\nu_{\pm}\ KLDY)$ (
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle$.
0
)
|
$|_{i=1}^{n}\, |_{j=1}^{n}\, |_{k=1}^{n}\, !\langle U_i, S_j, D_k\rangle$.
$(S_j, U_i, D_k;\ yArtifact_{ijk})$.
$\langle U_i, D_k, [S_j, yArtifact_{ijk}]\rangle$.
$(D_k, U_i;\ ycMess_{ijk})$.
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}$[at U_{ijk} orig D_{ijk}] in
0

|
$|_{i=0}^{n}\, |_{j=1}^{n}\, |_{k=0}^{n}\, !(U_i, S_j, D_k;\ )$.
$(\nu\, Artifact)$ (
$\langle S_j, U_i, [D_k, Artifact_{ijk}]\rangle$.
$(D_k, S_j, Artifact_{ijk};\ )$.
$\langle S_j, D_k, [U_i, KU_i]\rangle$.
0
)

|
$|_{i=0}^{n}\, |_{j=0}^{n}\, |_{k=1}^{n}\, !(U_i, D_k, S_j;\ zArtifact_{ijk})$.
$\langle D_k, S_j, zArtifact_{ijk}\rangle$.
$(S_j, D_k, U_i;\ zKU_i)$.
$(\nu\, Mess_{ijk})$ (
$\langle D_k, U_i, \{Mess_{ijk}\}_{zKU_i}$[at D_{ijk} dest U_{ijk}]$\rangle$.
0
)

)

F.5 LySa process for experiment 2

$(\nu_{\pm}\ CA)\,($
$(\nu_{i=1}^{n}\ KU_i)$
$(\nu_{\pm}\ KLDY)\,($
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\quad\quad \{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n}\,|_{j=1}^{n}\,|_{k=1}^{n}\,!TLSSInitC_A(U_i, S_j).$
$TLSSC_A(U_i, S_j, D_k).$
$TLSRC_A(S_j, U_i, D_k, yArtifact_{ijk}).$
$TLSSInitC_B(U_i, D_k).$
$TLSSC_B(U_i, D_k, [S_j, yArtifact_{ijk}]).$
$(D_k, U_i;\ ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}$[at U_{ijk} orig D_{ijk}] in
0

|
$|_{i=0}^{n}\,|_{j=1}^{n}\,|_{k=0}^{n}\,!TLSSInitS_A(U_i, S_j).$
$TLSRS_A(U_i, S_j, D_k, []).$
$(\nu\ Artifact)\,($
$TLSSS_A(S_j, U_i, [D_k, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_i]).$
0
)

|
$|_{i=0}^{n}\,|_{j=0}^{n}\,|_{k=1}^{n}\,!TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_i).$
$(\nu\ Mess_{ijk})\,($
$\langle D_k, U_i, \{Mess_{ijk}\}_{zKU_i}$[at D_{ijk} dest U_{ijk}]$\rangle.$
0
)

)

F.6 LySa process for experiment 3

$(\nu_{\pm}\ CA)\,($
$(\nu_{i=1}^{n}\ KU_i)$
$(\nu_{\pm}\ KLDY)\,($

$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\quad \{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n}\,|_{j=1}^{n}\,|_{k=1}^{n}\,!TLSSInitC_A(U_i, S_j).$
$TLSSC_A(U_i, S_j, D_k).$
$TLSRC_A(S_j, U_i, D_k, yArtifact_{ijk}).$
$TLSSInitC_B(U_i, D_k).$
$TLSSC_B(U_i, D_k, [S_j, yArtifact_{ijk}]).$
$TLSRC_B(D_k, U_i, [], ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}$[at U_{ijk} orig D_{ijk}] in
0

|
$|_{i=0}^{n}\,|_{j=1}^{n}\,|_{k=0}^{n}\,!TLSSInitS_A(U_i, S_j).$
$TLSRS_A(U_i, S_j, D_k, []).$
$(\nu\ Artifact)\,($
$TLSSS_A(S_j, U_i, [D_k, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_i]).$
0
)

|
$|_{i=0}^{n}\,|_{j=0}^{n}\,|_{k=1}^{n}\,!TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_i).$
$(\nu\ Mess_{ijk})\,($
$TLSSS_B(D_k, U_i, \{Mess_{ijk}\}_{zKU_i}$[at D_{ijk} dest U_{ijk}]).
0
)

)

F.7 LySa process for experiment 4

$(\nu_\pm\ CA)\,($
$(\nu_{i=1}^{n}\ KU_i)$
$(\nu_\pm\ KLDY)\,($
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\quad \{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
)
|

$|_{i=1}^{n} |_{j=1}^{n} |_{k=1}^{n} !TLSBInitC_A(U_i, S_j).$
$TLSSC_A(U_i, S_j, D_k).$
$TLSRC_A(S_j, U_i, D_k, yArtifact_{ijk}).$
$TLSSInitC_B(U_i, D_k).$
$TLSSC_B(U_i, D_k, [S_j, yArtifact_{ijk}]).$
$TLSRC_B(D_k, U_i, [], ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}$[at U_{ijk} orig D_{ijk}] in
0

|
$|_{i=0}^{n} |_{j=1}^{n} |_{k=0}^{n} !TLSBInitS_A(U_i, S_j).$
$TLSBRS_A(U_i, S_j, D_k, []).$
$(\nu\ Artifact)$ (
$TLSBSS_A(S_j, U_i, [D_k, Artifact_{ijk}]).$
$TLSSInitS_C(D_k, S_j).$
$TLSRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSSS_C(S_j, D_k, [U_i, KU_i]).$
0
)

|
$|_{i=0}^{n} |_{j=0}^{n} |_{k=1}^{n} !TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSSInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_i).$
$(\nu\ Mess_{ijk})$ (
$TLSSS_B(D_k, U_i, \{Mess_{ijk}\}_{zKU_i}$[at D_{ijk} dest U_{ijk}]).
0
)

)

F.8 LySa process for experiment 5

$(\nu_\pm\ CA)$ (
$(\nu_\pm\ KLDY)$ (
$\langle CA^+, KLDY^-, \{|LDY, KLDY^+|\}_{CA^-}, \{|U_0, KLDY^+|\}_{CA^-},$
$\quad \{|S_0, KLDY^+|\}_{CA^-}, \{|D_0, KLDY^+|\}_{CA^-}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n} |_{j=1}^{n} |_{k=1}^{n} !TLSBInitC_A(U_i, S_j).$
$TLSSC_A(U_i, S_j, D_k).$
$TLSRC_A(S_j, U_i, D_k, [yKU_{ijk}, yArtifact_{ijk}]).$
$TLSSInitC_B(U_i, D_k).$
$TLSSC_B(U_i, D_k, [S_j, yArtifact_{ijk}]).$
$TLSRC_B(D_k, U_i, [], ycMess_{ijk}).$

decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{yKU_{ijk}}$[at U_{ijk} orig D_{ijk}] in
0

|
$|_{i=0}^{n}\ |_{j=1}^{n}\ |_{k=0}^{n}\ !TLSBInitS_A(U_i, S_j).$
$TLSBRS_A(U_i, S_j, D_k, []).$
$(\nu\ Artifact_{ijk})$ (
$(\nu\ KU_{ijk})$ (
$TLSBSS_A(S_j, U_i, [D_k, KU_{ijk}, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_{ijk}]).$
0
)
)

|
$|_{i=0}^{n}\ |_{j=0}^{n}\ |_{k=1}^{n}\ !TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_{ijk}).$
$(\nu\ Mess_{ijk})$ (
$TLSSS_B(D_k, U_i, \{Mess_{ijk}\}_{zKU_{ijk}}$[at D_{ijk} dest U_{ijk}]).
0
)

)

F.9 LySa process for experiment 6

$(\nu_{\pm}\ CA)$ (
$(\nu_{\pm}\ KLDY)$ (
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n}\ |_{j=1}^{n}\ |_{k=1}^{n}\ !TLSBInitC_A(U_i, S_j).$
$TLSSC_A(U_i, S_j, D_k).$
$TLSRC_A(S_j, U_i, D_k, [yKU_{ijk}, yArtifact_{ijk}]).$
$TLSSInitC_B(U_i, D_k).$
$TLSSC_B(U_i, D_k, [S_j, yArtifact_{ijk}]).$
$(D_k, U_i;\ ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{yKU_{ijk}}$[at U_{ijk} orig D_{ijk}] in
0

|

$|_{i=0}^{n} |_{j=1}^{n} |_{k=0}^{n} !TLSBInitS_A(U_i, S_j).$
$TLSBRS_A(U_i, S_j, D_k, []).$
$(\nu\, Artifact_{ijk})\,($
$(\nu\, KU_{ijk})\,($
$TLSBSS_A(S_j, U_i, [D_k, KU_{ijk}, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_{ijk}]).$
0
)
)

|
$|_{i=0}^{n} |_{j=0}^{n} |_{k=1}^{n} !TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_{ijk}).$
$(\nu\, Mess_{ijk})\,($
$\langle D_k, U_i, \{Mess_{ijk}\}_{zKU_{ijk}}[\mathsf{at}\, D_{ijk}\ \mathsf{dest}\, U_{ijk}\,]\rangle.$
0
)

)

F.10 LySa process for experiment 7

$(\nu_\pm\ CA)\,($
$(\nu_{i=0}^{n}\ KU_i)($
$(\nu_\pm\ KLDY)\,($
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\quad \{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n} |_{j=1}^{n} |_{k=1}^{n} !\langle U_i, D_k\rangle.$
$(D_k, U_i;\ yD_{ijk}, yS_{ijk}).$
$TLSSInitC_A(U_i, yS_{ijk}).$
$TLSSC_A(U_i, yS_{ijk}, yD_{ijk}).$
$TLSRC_A(yS_{ijk}, U_i, yD_{ijk}, yArtifact_{ijk}).$
$TLSSInitC_B(U_i, yD_{ijk}).$
$TLSSC_B(U_i, yD_{ijk}, [yS_{ijk}, yArtifact_{ijk}]).$
$TLSRC_B(yD_{ijk}, U_i, [], ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}[\mathsf{at}\, U_{ijk}\ \mathsf{orig}\, D_{ijk}\,]$ in
0

|
$|_{i=0}^{n} |_{j=1}^{n} |_{k=1}^{n} !TLSSInitS_A(U_i, S_j).$

$TLSRS_A(U_i, S_j, D_k, []).$
$(\nu\, Artifact_{ijk})\,($
$TLSSS_A(S_j, U_i, [D_k, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_i]).$
0
)

|
$|_{i=0}^{n}\,|_{j=1}^{n}\,|_{k=1}^{n}\,!(U_i, D_k;\,).$
$\langle D_k, U_i, D_k, S_j\rangle.$
$TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_i).$
$(\nu\, Mess_{ijk})\,($
$TLSSS_B(D_k, U_i, \{Mess_{ijk}\}_{zKU_i}[\mathsf{at}\, D_{ijk}\ \mathsf{dest}\, U_{ijk}\,]).$
0
)

))

F.11 LySa process for experiment 8

$(\nu_{\pm}\ CA)\,($
$(\nu_{i=0}^{n}\ KU_i)($
$(\nu_{\pm}\ KLDY)\,($
$\langle CA^+, KLDY^-, \{\!|LDY, KLDY^+|\!\}_{CA^-}, \{\!|U_0, KLDY^+|\!\}_{CA^-},$
$\quad\{\!|S_0, KLDY^+|\!\}_{CA^-}, \{\!|D_0, KLDY^+|\!\}_{CA^-}, KU_0\rangle.$
0
)
|
$|_{i=1}^{n}\,|_{j=1}^{n}\,|_{k=1}^{n}\,!TLSSInitC_D(U_i, D_k).$
$TLSSC_D(U_i, D_k, []).$
$TLSRC_D(D_k, U_i, [], [yD_{ijk}, yS_{ijk}]).$
$TLSSInitC_A(U_i, yS_{ijk}).$
$TLSSC_A(U_i, yS_{ijk}, yD_{ijk}).$
$TLSRC_A(yS_{ijk}, U_i, yD_{ijk}, yArtifact_{ijk}).$
$TLSSInitC_B(U_i, yD_{ijk}).$
$TLSSC_B(U_i, yD_{ijk}, [yS_{ijk}, yArtifact_{ijk}]).$
$TLSRC_B(yD_{ijk}, U_i, [], ycMess_{ijk}).$
decrypt $ycMess_{ijk}$ as $\{;\ yMess_{ijk}\}_{KU_i}[\mathsf{at}\, U_{ijk}\ \mathsf{orig}\, D_{ijk}\,]$ in
0

|
$|_{i=0}^{n}\,|_{j=1}^{n}\,|_{k=1}^{n}\,!TLSSInitS_A(U_i, S_j).$

$TLSRS_A(U_i, S_j, D_k, []).$
$(\nu\, Artifact_{ijk})\, ($
$TLSSS_A(S_j, U_i, [D_k, Artifact_{ijk}]).$
$TLSBInitS_C(D_k, S_j).$
$TLSBRS_C(D_k, S_j, Artifact_{ijk}, []).$
$TLSBSS_C(S_j, D_k, [U_i, KU_i]).$
0
)

|
$|_{i=0}^{n} |_{j=1}^{n} |_{k=1}^{n}\, !TLSSInitS_D(U_i, D_k).$
$TLSRS_D(U_i, D_k, [], []).$
$TLSSS_D(D_k, U_i, [D_k, S_j]).$
$TLSSInitS_B(U_i, D_k).$
$TLSRS_B(U_i, D_k, [S_j], [zArtifact_{ijk}]).$
$TLSBInitC_C(D_k, S_j).$
$TLSSC_C(D_k, S_j, zArtifact_{ijk}).$
$TLSRC_C(S_j, D_k, U_i, zKU_i).$
$(\nu\, Mess_{ijk})\, ($
$TLSSS_B(D_k, U_i, \{Mess_{ijk}\}_{zKU_i}[\mathsf{at}\, D_{ijk}\ \mathsf{dest}\, U_{ijk}]).$
0
)

))

Appendix G

Test of CLySa Expansion

G.1 Alpha Conversion

Before Expansion:

define $Test(A, B) := \langle X, A, K^+, k^+, K^-, k^- \rangle$.
$(; x, B)$.
$\langle \{A, B\}_{k^-}$ [at A dest B], $\{\!|A, B|\!\}_k$[at A dest B]$\rangle$.
0

in

$(\nu_\pm\ k)$ (
$Test_{id}(a, b)$.
0
)

After Expansion:

$(\nu_\pm\ k)$ (
$\langle Xid, a, Kid^+, k^+, Kid^-, k^- \rangle$.
$(; xid, b)$.
$\langle \{a, b\}_{k^-}$ [at $Aid1$ dest $Bid1$], $\{\!|a, b|\!\}_k$[at $Aid2$ dest $Bid2$]$\rangle$.
0
)

G.2 Substitution

Before Expansion:

define $Test(A, B, C, D) :=$ decrypt A as $\{; B\}_{C^+}$[at UCP] in
decrypt A as $\{\!|; B|\!\}_{C^-}$[at UCP] in
$(x; y)$.
0

in

$Test_{id}(a, b, c, d).$
0

After Expansion:

decrypt a as $\{; b\}_c$[at UCP] in
decrypt a as $\{\!|; b|\!\}_c$[at UCP] in
$(xid;\ yid).$
0

G.3 Macro Insertion

Before Expansion:

define $Inside(F, G) := \langle F, G, x, k^+, zzz\rangle.$
0
,
$Test(A, B) := (\nu\, x)\,($
$(\nu_\pm\ k)\,($
$(A;\ x).$
$Inside_n(A, B).$
decrypt A as $\{B;\ z\}_{key}$[at UCP] in
decrypt B as $\{\!|A;\ z|\!\}_{k^+}$[at UCP] in
0
)
)

in

$(\nu\, key)\,($
$Test_{id}(C, D).$
0
)

After Expansion:

$(\nu\, key)\,($
$(\nu\, xid)\,($
$(\nu_\pm\ kid)\,($
$(C;\ xid).$
$\langle C, D, xnid, knid^+, zzznid\rangle.$
decrypt C as $\{D;\ zid\}_{key}$[at UCP] in
decrypt D as $\{\!|C;\ zid|\!\}_{kid^+}$[at UCP] in
0
)

)
)

G.4 Macro Expansion

Before Expansion:

define $Test(D, E) := \langle D, E, b, k \rangle$.
0

in

$(\nu^{n}_{\pm\, i=1}\ key)($
$(\nu^{n}_{j=1}\ name)($
$|^{n}_{i=1}\, !\langle A, B \rangle.$
$Test_{IDa}(A, B).$
0
|
$|^{n}_{j=1}\, |^{n}_{i=1\, i \neq j}\, !(\nu\, b)\,($
$(\nu_{\pm}\ k)\,($
$(A, B;\ x, y).$
decrypt x as $\{A, B;\ z\}_{key}$[at UCP] in
decrypt y as $\{\!|A, B;\ z|\!\}_{key}$[at UCP] in
$Test_{IDb}(A, B).$
0
)
)

))

After Expansion:

$(\nu^{n}_{\pm\, i=1}\ key)($
$(\nu^{n}_{j=1}\ name)($
$|^{n}_{i=1}\, !\langle A, B \rangle.$
$\langle A, B, bIDa_i, kIDa_i \rangle.$
0
|
$|^{n}_{j=1}\, |^{n}_{i=1\, i \neq j}\, !(\nu\, b)\,($
$(\nu_{\pm}\ k)\,($
$(A, B;\ x, y).$
decrypt x as $\{A, B;\ z\}_{key}$[at UCP] in
decrypt y as $\{\!|A, B;\ z|\!\}_{key}$[at UCP] in
$\langle A, B, b, k \rangle.$
0
)
)

))

G.5 List Expansion

Before Expansion:

$(\nu^{n}_{\pm\, i=1}\ key)($

$(\nu^{n}_{j=1}\ name)($

$|^{n}_{i=1}\, !\langle A, [A_i, A_i, A_i, []], [B], B\rangle.$

$\langle[\{[A], A\}_A[\mathsf{at}\, UCP\,]]\rangle.$

$\langle\{[A]\}_{\{A,B\}_{key}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]\rangle.$

$\langle[\{\!|[A], A|\!\}_A[\mathsf{at}\, UCP\,]]\rangle.$

$\langle\{\!|A|\!\}_{\{\!|A,B|\!\}_{key}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]\rangle.$

0

|

$|^{n}_{j=1}\, |^{n}_{i=1\, i\neq j}\, !(\nu\, b)\,($

$(\nu_{\pm}\ k)\,($

$([B_j, B_j, B_j], k^-, k^+, B;\ [x], [], y).$

decrypt A as $\{A, [B];\ x\}_{\{[A]\}_{key}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]$ in

decrypt A as $\{\!|A, [B];\ x|\!\}_{\{\!|[A]|\!\}_{k^-}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]$ in

0

)

)

))

After Expansion:

$(\nu^{n}_{\pm\, i=1\ldots n}\ key)$

$(\nu^{n}_{j=1\ldots n}\ name)$

$|^{n}_{i=1\ldots n}\, !\langle A, A_i, A_i, A_i, B, B\rangle.$

$\langle\{A, A\}_A[\mathsf{at}\, UCP\,]\rangle.$

$\langle\{A\}_{\{A,B\}_{key}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]\rangle.$

$\langle\{\!|A, A|\!\}_A[\mathsf{at}\, UCP\,]\rangle.$

$\langle\{\!|A|\!\}_{\{\!|A,B|\!\}_{key}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]\rangle.$

0

|

$|^{n}_{j=1\ldots n}\, |^{n}_{i=1\ldots n\, i\neq j}\, !(\nu\, b)\,(\nu_{\pm}\ k)\,(B_j, B_j, B_j, k^-, k^+, B;\ x, y).$

decrypt A as $\{A, B;\ x\}_{\{A\}_{key}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]$ in

decrypt A as $\{\!|A, B;\ x|\!\}_{\{\!|A|\!\}_{k^-}[\mathsf{at}\, UCP\,]}[\mathsf{at}\, UCP\,]$ in

0

G.6 Numbering

Before Expansion:

$(\nu^{n}_{i=1\, j=1}\ K_{ij})($

$(\nu^{n}_{\pm\, i=1\, j=1}\ H_{ij})($

$|^{n}_{i=1}\ (\nu\, \#x)\,($

$\langle\{\#A, \#B\}_{\#k}[\mathsf{at}\, \#A\ \mathsf{dest}\, \#B\,], \#B\rangle.$

$(\#A;\ \#x)$.
decrypt $\#x$ as $\{\#A, \#B;\ \#x\}_{\#k}$[at $\#A$ orig $\#B$] in
0
)
|
$|_{j=1}^{n}$ $(\nu_{\pm}\ \#x)$ (
$\langle \#A, \#B \rangle$.
$(\{\!|\#A, \#B|\!\}_{\#k}$[at $\#A$ dest $\#B$]; $\#x)$.
decrypt $\#x$ as $\{\!|\#A, \#B;\ \#x|\!\}_{\#k}$[at $\#A$ orig $\#B$] in
0
)

))

After Expansion:

$(\nu_{i=1\,j=1}^{n}\ K_{ij})($
$(\nu_{\pm\,i=1\,j=1}^{n}\ H_{ij})($
$|_{i=1}^{n}$ $(\nu\, x1)$ (
$\langle \{A1, B1\}_{k1}$[at $A1$ dest $B1$]$, B2\rangle$.
$(A2;\ x2)$.
decrypt $x3$ as $\{A3, B3;\ x4\}_{k2}$[at $A2$ orig $B2$] in
0
)
|
$|_{j=1}^{n}$ $(\nu_{\pm}\ x1)$ (
$\langle A1, B1\rangle$.
$(\{\!|A2, B2|\!\}_{k1}$[at $A1$ dest $B1$]; $x2)$.
decrypt $x3$ as $\{\!|A3, B3;\ x4|\!\}_{k2}$[at $A2$ orig $B2$] in
0
)

))

Bibliography

[1] D. Eastlake 3rd and P. Jones. US secure hash algorithm 1 (SHA1). RFC 3174, Internet Engineering Task Force, September 2001.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *Software Engineering, IEEE Transactions on*, 22(1):6–15, 1996.

[3] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[4] Paul C. Kocher Alan O. Freier, Philip Karlton. *The SSL Protocol*, version 3.0 draft edition, 1996. `http://wp.netscape.com/eng/ssl3/draft302.txt`.

[5] R. Anderson and R. Needham. Robustness principles for public key protocols. *Advances in Cryptology - CRYPTO '95. 15th Annual International Cryptology Conference. Proceedings*, pages 236–47, 1995.

[6] G. Bella, F. Massacci, and L.C. Paulson. Verifying the set registration protocols. *Selected Areas in Communications, IEEE Journal on*, 21(1):77–87, 2003.

[7] Karthikeyan Bhargavan, Cèdric Fournet, and Andrew D. Gordon. Verifying policy-based security for web services. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 268–277. ACM Press, 2004.

[8] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th Computer Security Foundations Workshop (CSFW 03).*, pages 126–140. IEEE Computer Society Press, 2003.

[9] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

[10] P. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. *Computer Security Foundations Workshop, 2003. Proceedings. 16th IEEE*, pages 141–151, 2003.

[11] M. Buchholtz, F. Nielson, and H. Riis Nielson. *Static Analysers*, February 2004. `http://www.imm.dtu.dk/cs_LySa/d19-1.1.pdf`.

[12] M. Burrows, M. Abad, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 426(1871):233–271, 1989.

[13] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, Internet Engineering Task Force, January 1999.

[14] D. Dolev and A.C. Yao. On the security of public key protocols. *22nd Annual Symposium on Foundations of Computer Science*, pages 350–7, 1981.

[15] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[16] C. Fischer and H. Wehrheim. Model-checking csp-oz specifications with fdr. *IFM'99. Proceedings of the 1st International Conference on Integrated Formal Methods*, pages 315–34, 1999.

[17] T. Gross. Security analysis of the saml single sign-on browser/artifact profile. *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 298–307, 2003.

[18] S. Hansen, J. Skriver, and H. Riis Nielson. Using static analysis to validate the saml single sign-on protocol. In *Proceedings of Workshop on Issues in the Theory of Security (WITS'05)*, to appear in 2005.

[19] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[20] John Hughes and Eve Maler. *Tecnical Overview of the OASIS Security Assertion Markup Language*, version 1.1 edition, May 2004. `http://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf`.

[21] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: keyed-hashing for message authentication. RFC 2104, Internet Engineering Task Force, February 1997.

[22] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. *Software-Concepts and Tools*, 17(3):93–102, 1996.

[23] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[24] Eve Maler, Prateek Mishra, and Rob Philpott. *Assertions and Protocol for the OASIS Security Assertion Markup Language*, version 1.1 edition, May 2003. `http://xml.coverpages.org/SAML-v11-cs01-core.pdf`.

[25] Eve Maler, Prateek Mishra, and Robert Philpott. *Bindings and Profiles for the OASIS Security Assertion Markup Language*, version 1.1 edition, May 2003. `http://xml.coverpages.org/SAML-v11-cs01-bindings.pdf`.

[26] Eve Maler and Rob Philpott. *Security and pricacy Considerations for the OASIS Security Assertion Markup Language*, version 1.1 edition, May 2003. `http://xml.coverpages.org/SAML-v11-cs01-secConsider.pdf`.

[27] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. i. *Information and Computation*, 100(1):1–40, 1992.

[28] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. ii. *Information and Computation*, 100(1):41–77, 1992.

[29] Nilo Mitra. Soap version 1.2 part 0: Primer. W3C working draft, World Wide Web Consortium (W3C), December 2001.

[30] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–9, 1978.

[31] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer-Verlag New York, Inc., 1999.

[32] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. A succinct solver for alfp. *Nordic J. of Computing*, 9(4):335–372, 2002.

[33] H. Riis Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction.* Wiley, 1992.

[34] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

[35] R. Rivest. The MD5 message-digest algorithm. RFC 1321, Internet Engineering Task Force, April 1992.

[36] *Common Criteria.* `http://www.commoncriteriaportal.org/`.

[37] *Liberty Alliance project.* `http://projectliberty.org/`.

[38] LySa-*implementation.* `http://www.imm.dtu.dk/cs_LySa/`.

[39] *Succinct Solver implementation.* `http://www.imm.dtu.dk/cs_SuccinctSolver/`.

[40] *Standard ML of New Jersey (SML/NJ).* `http://www.smlnj.org/`.

[41] *Organization for the Advancement of Structured Information Standards.* `http://www.oasis-open.org/`.

[42] *Shibboleth project.* `http://shibboleth.internet2.edu/index.html`.

[43] *Workshop on Issues in the Theory of Security (WITS'05).* `http://chacs.nrl.navy.mil/projects/wits05/`.