CaPiTo: protocol stacks for services

Han Gao, Flemming Nielson, Hanne Riis Nielson

Under consideration for publication in Formal Aspects of Computing

CaPiTo: Protocol Stacks for Services

Han Gao[1] and Flemming Nielson[1] and Hanne Riis Nielson[1]

[1]DTU Informatics, Technical University of Denmark

Abstract. CaPiTo allows the modelling of service-oriented applications using process algebras at three levels of abstraction. The *abstract* level focuses on the key functionality of the services; the *plug-in* level shows how to obtain security using standardised protocol stacks; finally, the *concrete* level allows to consider how security is obtained using asymmetric and symmetric cryptographic primitives.

The CaPiTo approach therefore caters for a variety of developers that need to cooperate on designing and implementing service-oriented applications. We show how to formally analyse CaPiTo specifications for ensuring the absence of security flaws. The method used is based on static analysis of the corresponding LySa specifications. We illustrate the development on two industrial case studies; one taken from the banking sector and the other a single sign-on protocol.

1. Introduction

Service-oriented systems are becoming the cutting edge of IT Systems. In this context, services are viewed as independent computational entities that can be described, published, discovered, etc. The most common implementation of service-oriented systems is protocol-based web services, where security of interactions between clients and services is ensured by means of industrial communication protocols, e.g. *TLS* [DA99], *SOAP* [SOA], *HTTP*, *TCP/IP*.

Different levels of abstractions are employed by the researchers, developers and programmers participating in the design of service-oriented applications. The CaPiTo approach (presented in Section 2) aims at bridging these gaps within one uniform framework.

- At the *abstract* level of CaPiTo the functionality of the system is designed without particular attention to the methods needed for ensuring security. This level is useful for developing the overall functionality of the system and for ensuring that services operate as intended.
- At the *plug-in* level of CaPiTo existing security protocol stacks are attached to the abstract specifications so as to clarify which methods should be used for ensuring security. We consider it a main achievement of CaPiTo that we are able to incorporate existing protocols such as *TLS* and *SOAP*. This level is useful

Correspondence and offprint requests to: DTU Informatics, Technical University of Denmark. E-mail: han.22cc@gmail.com, nielson@imm.dtu.dk, riis@imm.dtu.dk

for interacting with software developers and for cross-compilation into the actual programming language of interest, e.g. C or Java.

- At the *concrete* level the security protocols are expanded out into their underlying symmetric or asymmetric cryptographic primitives. This is in many ways the level that is closest to the traditional Alice Bob protocol narrations, but here formulated in the form of processes in a process algebra. This level is useful for detailed analyses of the security of the cryptographic protocols embedded in the service-oriented applications.

We provide (in Section 3) two case studies of using this approach. One is the financial Credit Request case study taken from the SENSORIA project [Sen]. The other is the SAML Single Sign-On protocol case study from the Liberty-Alliance project [Lib]. We show how one can benefit from the different levels of modelling in the case studies.

We then develop (in Section 4) a static analysis for ensuring the absence of security flaws in the CaPiTo concrete level specification. This approach is based on our LySa approach [BBD$^+$05] and aims at providing a safe over-approximation of the protocol behaviour.

While we have performed code generation from the CaPiTo plug-in level specification into C, using a library of encoded protocols, this is not part of the present paper.

Related Work. Researchers have developed many general purpose verification tools intended to be used for modeling and verifying cryptographic protocols. Many of these focus on a process algebraic way of expressing the Alice Bob narrations typically used when expressing cryptographic protocols. As an example, work was conducted by using LOTOS [LG00] (based on CSP) to specify the protocols and the FDR model checker was used to establish trace inclusion between the system and the property. Abadi and Gordon [AG99] extended the π-calculus with cryptographic primitives (e.g. keys, nonces, encryptions, signatures) to obtain their spi-calculus that admitted cryptographic terms into the processes. This approach proves a protocol secure by establishing bisimulation equivalences between two processes.

Our own work on LySa [BBD$^+$05] was inspired by the spi-calculus but used methods from static analysis to analyse security of the protocols. Developments with a somewhat similar goal also include ProVerif [Pro], Scyther [Scy], and OFMC [MV09]; this list is not intended to be complete!

Recently, process algebras have been used to define clean semantics models for service-oriented systems. This trend is witnessed by the many process calculi-like formalisms for client-server interaction, orchestration and the handling of unexpected events. Example process algebras include COWS [LPT07], with a focus on service orchestration, and CaSPiS [BBNL08], where the notions of session and pipelining play a central role.

The work presented in this paper goes one step further in modelling service oriented systems at different levels of abstraction and therefore tries to bridge the gap between researchers, developers and programmers. Previous ideas in this direction were presented in [GNN11].

2. The CaPiTo Approach to Services

The CaPiTo approach to modelling service-oriented applications involves a total of three levels. At the *abstract* level, we abstract away from both cryptography and standardised communication protocols, and this allows us to concentrate on the interaction of the services themselves. At the *plug-in* level the abstract specification is augmented with plug-ins for identifying the standardised protocol stacks to be used but without going into the details of the cryptography. At the *concrete* level, the notion of *services* is hidden, while we model both cryptography and the details of the standardised communication protocols used for implementing the service-oriented application.

The transformation from the *abstract* level to the *plug-in* level needs to be performed manually because it is

Table 1. The abstract level of CaPiTo.

$$\begin{array}{lll}
v, w & ::= & n \mid f(\vec{v}) \\
e & ::= & x \mid n \mid f(\vec{e}) \\
p & ::= & ?x \mid x \mid n \\
P & ::= & \langle \vec{e} \rangle.P \mid (\vec{p}).P \mid (\nu\ n)P \mid P_1|P_2 \mid !P \mid P_1 + P_2 \mid 0 \mid \overline{n\nu}[\,].P \mid n\nu[\,].P \mid \ \uparrow\langle \vec{e} \rangle.P
\end{array}$$

here that one makes the design decision of choosing between the many different protocol suites that might be appropriate. The unfolding of sessions at the *plug-in* level and the transformation to the *concrete* level, however, is fully automatic. We do not consider the semantic correctness of unfoldings and transformations but merely devise the formal semantics at the *concrete* level.

2.1. The Abstract Level of CaPiTo

In the abstract modelling of service-oriented applications, certain details are often abstracted away, for example, the underlying standardised protocols to be used to protect the communication steps. The abstract level of CaPiTo incorporates this abstraction.

The basic building blocks are values, $v, w \in Val$, which correspond to *closed* expressions, i.e. expressions without free variables. Values are used to represent keys, nonces, messages, functions, etc. Syntactically, they are described by expressions $e \in Expr$. We will use n to range over names, x to range over variables, and e to range over expressions. We use $\vec{v}$ as a short-hand for $v_1, \ldots, v_k$. In addition, we allow the use of functions f (on top of the constructors and destructors that will be used in the other levels of CaPiTo for modelling cryptography).

Communication is facilitated by means of pattern matching and the binding of values to variables. We distinguish between *defining* occurrences and *applied* occurrences of variables. A defining occurrence is an occurrence where the binding of a variable is performed, while an applied occurrence is an occurrence of a variable where the value bound to it is being requested. We perform this distinction in a clear syntactic way: the defining occurrence of a variable x is denoted by $?x$, while in the scope of the declaration, the applied occurrences appear as x. For example, the communication from the output action $\langle A \rangle.0$ to the input action $(?x).P$ succeeds and results in all the occurrences of the variable x in P being replaced with the value A. Some calculi do not perform a clear syntactic distinction between defining and applied occurrences of variables; they either do not permit pattern matching, e.g. the π-calculus [Mil99], or require the defining occurrences in a matching to be separated from the applied occurrences by some symbol, e.g. the use of the semi-colon in LySa [BBD$^+$05]. Our approach is more general while still allowing us to define a useful static analysis in Section 4.

The syntax of the abstract specification is in Table 1. Services are syntactically built from basic activities, including the invoke activity ($\overline{n\nu}[\,].P$), the receive activity ($n\nu[\,].P$), where $n\nu[\,].P$ defines a service that can be invoked by $\overline{n\nu}[\,].P$. Other activities include restriction ($(\nu\ n)P$), nondeterministic choice ($P_1 + P_2$), parallel composition ($P_1|P_2$), replication ($!P$), and service return ($\uparrow \langle \vec{e} \rangle.P$), where values $\vec{e}$ are returned outside a service to the enclosing environment. In $\overline{n_1\nu}[\,].(\overline{n_2\nu}[\,].((A, ?x). \uparrow \langle x \rangle.0))$, the process $\uparrow \langle x \rangle$ sends $\langle x \rangle$ to the environment, thereby bypassing n_2 and making it an output belonging to service n_1.

After a service has been invoked there is a sequence of communication steps taking place. The empty holes [ ] after the service invocations and responses (e.g. $\overline{req}$ and req) serve as place-holders for the *plug-in* level to be presented below: to provide a list of the underlying security protocols to be used to protect the communication steps. The reader familiar with [BBNL08] will notice that our abstract level has been inspired by the CaSPiS process calculus developed in the SENSORIA project [Sen].

Example 1. Let us consider a simple scenario where a client asks his or her bank for the account balance as

Table 2. The plug-in level of CaPiTo: still using sessions.

$$
\begin{array}{lll}
v, w & ::= & n \mid f(\vec{v}) \\
e & ::= & x \mid n \mid f(\vec{e}) \\
p & ::= & ?x \mid x \mid n \\
P & ::= & \langle \vec{e} \rangle.P \mid (\vec{p}).P \mid (\nu\, n)P \mid P_1|P_2 \mid !P \mid P_1{+}P_2 \mid 0 \mid \overline{n\nu}[ps].P \mid n\nu[ps].P \mid \ \uparrow\langle \vec{e} \rangle.P \\
ps & ::= & pi \mid pi; ps \\
pi & ::= & name, param_1, \cdots, param_k
\end{array}
$$

captured by the following protocol narration:

$$
\begin{array}{llll}
Client & \rightarrow & Bank: & TS \\
Bank & \rightarrow & Client: & TS, Bal
\end{array}
$$

In the first message the client sends a time stamp (TS) to the bank and in the second message the bank replies with the account balance (Bal) and includes the time stamp so that the client will know that is an answer to his recent request.

In the service-oriented setup of CaPiTo, the bank will provide a *service* denoted $req\nu[\,]$ (for request) and it is invoked by the construct $\overline{req\nu}[\,]$. The client and the bank can now be specified as follows:

$$
\begin{array}{lll}
Client & \triangleq & (\nu\ TS)\ \overline{req\nu}[\,].\ \langle Client, Bank, TS \rangle. \\
 & & \qquad\qquad\quad (Bank, Client, TS, ?bal).0 \\
Bank & \triangleq & req\nu[\,].\ (Client, Bank, ?ts). \\
 & & \qquad\quad (\nu\ Bal)\langle Bank, Client, ts, Bal \rangle.0
\end{array}
$$

The overall system is the parallel composition of the above processes as given by

$$System \triangleq Client \mid Bank$$

Below we shall extend this example to be more specific about the underlying protocols and the establishment of the security of the message transfer. □

2.2. The Plug-In Level of CaPiTo

From the point of view of the OSI model [OSI], messages have to pass several layers before reaching the physical link layer and actually being transmitted. Each layer receives information from the layer above, modifies it and passes it down to the next layer. The operation of each layer is controlled by different protocols — hence the layers are said to constitute a protocol stack. So far these considerations are not visible in the syntax of CaPiTo.

We now introduce an intermediate plug-in level in CaPiTo where we can retain part of the abstract view of service-oriented applications but where we can also begin to be precise about which protocols we intend to use. This takes the form of providing a list of protocols to be placed in the place-holders mentioned above and as shown in Table 2. Here ps is a protocol stack, which we take to be a non-empty list (separated by semi-colons) of the protocols to be used. Each protocol pi is identified by its name and a number of auxiliary parameters ($k \geq 0$).

*Example 2.*Let us extend Example 2.1 to use the TLS (Transport Layer Security) protocol [DA99] to protect the communication between the client and the bank.

The unilateral TLS protocol is summarised by the following protocol narration taking place between a client

C and a server S holding a certificate $\mathbf{S}_{K^-_{CA}}(S, K^+_S)$ issued by a mutually trusted Certificate Authority CA:

$$\begin{array}{lcccl}
1. & C & \rightarrow & S & : & N_C \\
2. & S & \rightarrow & C & : & N_S, \mathbf{S}_{K^-_{CA}}(S, K^+_S) \\
3. & C & \rightarrow & S & : & \mathbf{P}_{K^+_S}(N) \\
4. & C & \rightarrow & S & : & \mathbf{E}_{\mathbf{H}(N_C, N_S, N)}(M)
\end{array}$$

It is assumed that both principals know the public key K^+_{CA} of the certificate authority and a signature with the corresponding private key is denoted $\mathbf{S}_{K^-_{CA}}()$. Encryption with a public key K^+_S is denoted $\mathbf{P}_{K^+_S}()$ The certificate is then used to prove the identity of the server to the client so that a common master key can be agreed upon; this key is obtained as the hash $\mathbf{H}(N_C, N_S, N)$ of the three nonces N_C, N_S and N. All further messages are encrypted using this key using symmetric cryptography; this is denoted $\mathbf{E}_{\mathbf{H}(N_C, N_S, N)}()$.

The plug-in specification is obtained from the abstract specification of Example 2.1 by recording that the *TLS* protocol is to be used for protecting the communication, where *Client* acts as the initiator and *Bank* as the responder. The asymmetric key pair associated with *Bank* is $K^{\pm}_{Bank}$(for (K^+_{Bank}, K^-_{Bank})). The construct $(\nu_{\pm}\ K)$ is used to generate the new public/private key pair. The overall system will be specified as follows

$$\mathit{System} \triangleq (\nu_{\pm}\ K_{CA})(\nu_{\pm}\ K_{Bank})\ (\mathit{Client} \mid \mathit{Bank})$$

where the previous definitions of *Client* and *Bank* are modified by "plugging in" the parameters to the service invocations and responses as shown below

$$\begin{array}{lcl}
\mathit{Client} & \triangleq & (\nu\ TS)\ \ \overline{\mathit{req}\nu}[TLS, C, B, CA]. \\
& & \qquad\qquad \langle \mathit{Client}, \mathit{Bank}, TS\rangle. \\
& & \qquad\qquad (\mathit{Bank}, \mathit{Client}, TS, ?\mathit{bal}).0 \\
\mathit{Bank} & \triangleq & \ \ \mathit{req}\nu[TLS, C, B, CA]. \\
& & \qquad (\mathit{Client}, \mathit{Bank}, ?ts). \\
& & \qquad (\nu\ \mathit{Bal})\langle \mathit{Bank}, \mathit{Client}, ts, \mathit{Bal}\rangle.0
\end{array}$$

where C, B and CA stand for $C = \mathit{Client}$, $B = (\mathit{Bank}, K^{\pm}_{Bank})$ and $CA = K^{\pm}_{CA}$.

The use of the unilateral *TLS* protocol ensures that the server is authenticated to the client; however, it does not ensure that the client is authenticated to the server. This may be rectified by using the bilateral version of *TLS*. It is very similar to the unilateral version but extends the third message in the protocol narration to include the client's certificate as shown in:

$$3.\ C\ \rightarrow\ S\ :\ \mathbf{S}_{K^-_{CA}}(C, K^+_C), \mathbf{P}_{K^+_S}(N), \mathbf{S}_{K^-_C}(N_C, N_S, N)$$

To indicate the use of the bilateral version of *TLS* at the CaPiTo-level we simply change the plug-ins to use TLS_2 rather than *TLS*. The overall system is now specified as

$$\mathit{System} \triangleq (\nu_{\pm}\ K_{CA})(\nu_{\pm}\ K_{Client})(\nu_{\pm}\ K_{Bank})\ (\mathit{Client} \mid \mathit{Bank})$$

$$\begin{array}{lcl}
\mathit{Client} & \triangleq & (\nu\ TS)\ \ \overline{\mathit{req}\nu}[TLS_2, C, B, CA]. \\
& & \qquad\qquad \langle \mathit{Client}, \mathit{Bank}, TS\rangle. \\
& & \qquad\qquad (\mathit{Bank}, \mathit{Client}, TS, ?\mathit{bal}).0 \\
\mathit{Bank} & \triangleq & \ \ \mathit{req}\nu[TLS_2, C, B, CA]. \\
& & \qquad (\mathit{Client}, \mathit{Bank}, ?ts). \\
& & \qquad (\nu\ \mathit{Bal})\langle \mathit{Bank}, \mathit{Client}, ts, \mathit{Bal}\rangle.0
\end{array}$$

where C is now a short-hand defined by $C = (\mathit{Client}, K^{\pm}_{Client})$ and B and CA are defined as before. Clearly this requires a mechanism for explaining the detailed operation of the plug-in; we shall do so when considering the translation to the concrete level of CaPiTo. □

Unfolding sessions and protocol stacks. The notion of service is still too abstract for the plug-in level to serve as a useful specification language for how to incorporate standardised protocols. We therefore show

Table 3. The plug-in level of CaPiTo: now using session identifiers. (Only changes to Table 2 are shown.)

$$P \quad ::= \quad \langle r, \vec{e}\rangle.P \mid (r, \vec{p}).P \mid (\nu\ n)P \mid P_1|P_2 \mid !P \mid P_1 + P_2 \mid 0 \mid \overline{r}[pi] \rhd P \mid r[pi] \rhd P$$

how to replace the sessions with session identifiers that more directly control where and how fresh nonces are to be created in order to ensure the correct correlation between service invocation and services response. At the same time we unfold the protocol stacks to apply each protocol one by one. It is convenient to represent this as a "source-to-source" transformation at the plug-in level: transforming from plug-in with implicit *sessions* (Table 2) to plug-ins with explicit *session identifiers* (Table 3).

The main task is to distinguish between different services as well as different sessions of each service by using unique session identifiers. This is taken care of by a transformation function $\mathcal{T}$. The transformation function takes two arguments; the first one is a plug-in specification to be transformed and the second one is a stack for recording all the session identifiers that are generated along the way, with the topmost identifier as the most recent one. Initially the function is called with r_{env}, denoting the *environment*, as the second argument, i.e. $\mathcal{T}(P, [r_{env}])$. The result of applying the transformation function is a plug-in specification with explicit session identifiers, of which the syntax is as shown in Table 3. The auxiliary task is to unfold a protocol stack that is used by each service. When a protocol stack contains a sequence of protocols we first expand away the leftmost (topmost) protocol in the stack and then continue with the subsequent layers. The function $\mathcal{T}$ is defined as follows:

$$\begin{aligned}
\mathcal{T}(\overline{s\nu}[pi_1, \cdots, pi_k].P, rl) &\triangleq (\nu\ r)\langle r_{env}, s, r\rangle.\overline{r}[pi_k] \rhd (\cdots \overline{r}[pi_1] \rhd \mathcal{T}(P, r :: rl))\\
\mathcal{T}(s\nu[pi_1, \cdots, pi_k].P, rl) &\triangleq (r_{env}, s, ?r).r[pi_k] \rhd (\cdots r[pi_1] \rhd \mathcal{T}(P, r :: rl))\\
\mathcal{T}(\langle\vec{e}\rangle.P, r :: rl) &\triangleq \langle r, \vec{e}\rangle.\mathcal{T}(P, r :: rl)\\
\mathcal{T}((\vec{p}).P, r :: rl) &\triangleq (r, \vec{p}).\mathcal{T}(P, r :: rl)\\
\mathcal{T}(\uparrow\langle\vec{e}\rangle.P, r_1 :: r_2 :: rl) &\triangleq \langle r_2, \vec{e}\rangle.\mathcal{T}(P, r_1 :: r_2 :: rl)\\
\mathcal{T}((\nu\ n)P, rl) &\triangleq (\nu\ n)\mathcal{T}(P, rl)\\
\mathcal{T}(P_1|P_2, rl) &\triangleq \mathcal{T}(P_1, rl)|\mathcal{T}(P_2, rl)\\
\mathcal{T}(P_1 + P_2, rl) &\triangleq \mathcal{T}(P_1, rl) + \mathcal{T}(P_2, rl)\\
\mathcal{T}(!P, rl) &\triangleq !\mathcal{T}(P, rl)
\end{aligned}$$

Each service invocation $\overline{s\nu}[ps].P$ leads to the creation of a new session, identified by a fresh session identifier r, that is sent to the corresponding service $s\nu[ps].Q$ for synchronisation. At both invocation and responding sides, the identifier r is systematically attached to each communication belonging to the session, which imposes all the outputs $\langle\vec{e}\rangle$ and inputs $(\vec{p})$ to take the form $\langle r, \vec{e}\rangle$ and $(r, \vec{p})$, respectively. Since client and service may be far apart, a session naturally comes with two parties, written $\overline{r}[ps] \rhd P$ and $r[ps] \rhd P$, with r bound somewhere above them by $(\nu\ r)$.

In the first two lines of the definition of the transformation function, the messages for the communication of r use the global session identifier r_{env}, i.e. $\langle r_{env}, s, r\rangle$ and $(r_{env}, s, ?r)$. We do this to ensure that a search for matching invocation and service provider is performed in the entire system; it also ensures that outputs and inputs have the required form.

The session identifer r is used to correlate communications from the same service invocation and response. Multiple invocations to services will yield separate session identifiers, as identified by different r's, and the hierarchy of nested sessions is reflected by the position of each session identifer r in the identifier stack. In fact, when values are returned outside the current session using the return operator, $\uparrow\langle\vec{e}\rangle$, the second topmost identifer in the stack is adopted, which gives $\langle r_2, \vec{e}\rangle$ (see the fifth line of the function definition). As a fresh session identifier r is generated at each service invocation, it ensures that in the replication case, say, $!(\overline{s\nu}[ps] \ldots \mid s\nu[ps] \ldots)$, a unique session identifier is attached to each process copy.

Table 4. The concrete level of CaPiTo.

$$\begin{array}{lll}
v, w & ::= & n \mid n^{+} \mid n^{-} \mid \mathbf{P}_{n^{+}}(\vec{v}) \mid \mathbf{S}_{n^{-}}(\vec{v}) \mid \mathbf{H}(\vec{v}) \mid f(\vec{v}) \\
e & ::= & x \mid n \mid n^{+} \mid n^{-} \mid \mathbf{P}_{n^{+}}(\vec{e}) \mid \mathbf{S}_{n^{-}}(\vec{e}) \mid \mathbf{H}(\vec{e}) \mid f(\vec{e}) \\
p & ::= & ?x \mid x \mid n \mid n^{+} \mid n^{-} \mid \underline{\mathbf{P}}_{n^{-}}(\vec{p}) \mid \underline{\mathbf{S}}_{n^{+}}(\vec{p}) \\
P & ::= & \langle r, \vec{e} \rangle.P \mid (r, \vec{p}).P \mid (\nu\ n)P \mid !P \mid r : e \blacktriangleright P \mid (\nu_{\pm}\ n)P \mid P_1 + P_2 \mid P_1|P_2 \mid 0
\end{array}$$

Example 3. Consider a simple process P

$$\begin{array}{l}
\overline{s_1 \nu}[pi_1].(\\
\quad \overline{s_2 \nu}[pi_2].(\\
\quad\quad \langle A, B \rangle. \uparrow \langle A \rangle.0))
\end{array}$$

Applying the transformation function on P, e.g. $\mathcal{T}(P, [r_{env}])$, gives the following result,

$$\begin{array}{l}
(\nu\ r_1)\langle r_{env}, s_1, r_1 \rangle.\overline{r_1}[pi_1] \rhd (\\
\quad (\nu\ r_2)\langle r_{env}, s_2, r_2 \rangle.\overline{r_2}[pi_2] \rhd (\\
\quad\quad \langle r_2, A, B \rangle.\langle r_1, A \rangle.0))
\end{array}$$

After the transformation, service s_1 is identified by r_1 and s_2 by r_2. The communication $\langle A, B \rangle$ belongs to the session r_2 hence we have $\langle r_2, A, B \rangle$. The last message $\uparrow \langle A \rangle$ returns the value A to the session outside r_2, i.e. r_1, and therefore it is transformed into $\langle r_1, A \rangle$.

2.3. The Concrete Level of CaPiTo

At the concrete level of CaPiTo we fully model communication protocols and the use of asymmetric and symmetric cryptography. We write $\mathbf{P}_{v_0^{+}}(\vec{v})$ for asymmetric encryption and $\mathbf{S}_{v_0^{-}}(\vec{v})$ for digital signatures; we write $\underline{\mathbf{P}}_{v_0^{-}}(\vec{p})$ for asymmetric decryption and $\underline{\mathbf{S}}_{v_0^{+}}(\vec{p})$ for the validation of digital signatures; and we write $\mathbf{H}(\vec{v})$ for producing hash values of $\vec{v}$. Symmetric cryptography is modelled as communications through an encrypted tunnel, e.g. $r : e \blacktriangleright P$, with r being the session identifier and e being the symmetric key shared between the sending and receiving principals involved in the communications within P.

An alternative design decision would be to have encryptions instead of tunnels, for example $\{A, B\}_k$ instead of $r : k \blacktriangleright \langle A, B \rangle$, as is often found in other calculi, for example, the spi-calculus, as well as in Alice Bob protocol narrations. In our view, tunnels are one of the basic building blocks in real-life protocols, e.g. IPSec Architecture [Sta99], and being able to accommodate it in a process algebra shows their power and convenience.

Expanding protocol behaviour. Services described at the plug-in level of CaPiTo contain the names of the protocols that need to be applied. The formal definition of these protocols is made clear at the concrete level and corresponds to expanding the behaviour of protocols; each protocol gives rise to a set of expansion rules.

The unilateral *TLS* protocol is defined in Table 5 and the bilateral *TLS* protocol is defined in Table 6 and are parameterised on the name of the service to be protected by the protocol. The definition is explicit about the creation of nonces and the checking of certificates (as explained informally in the protocol narrations of Example 2.2). After the completion of the the handshakes, the symmetric key $\mathbf{H}(N_C, N_S, N)$ is computed and used to tunnel the remainder of the communication steps. Note that the session identifer r is attached to each input and output in order to achieve the desired correlation.

Table 5. Unilateral *TLS* protocol for the service between C and S, where $S' = (S, K_S^{\pm})$ and $CA = K_{CA}^{\pm}$.

$$
\begin{array}{lcl}
\overline{r}[\mathit{TLS}, C, S', \mathit{CA}] \rhd (\langle r, \vec{v}\rangle.P) & \triangleq & (\nu\ N_C)\langle r, C, S, N_C\rangle. \\
 & & \quad (r, S, C, ?n_S, \underline{\mathbf{S}}_{K_{CA}^{+}}(S, ?x_{ks})). \\
 & & \quad\quad (\nu\ N)\langle r, C, S, \mathbf{P}_{x_{ks}}(N)\rangle. \\
 & & \quad\quad\quad r : \mathbf{H}(N_C, n_S, N) \blacktriangleright (\langle r, \vec{v}\rangle.P) \\
r[\mathit{TLS}, C, S', \mathit{CA}] \rhd ((r, \vec{p}).P) & \triangleq & (r, C, S, ?n_C). \\
 & & \quad (\nu\ N_S)\langle r, S, C, N_S, \mathbf{S}_{K_{CA}^{-}}(S, K_S^{+})\rangle. \\
 & & \quad\quad (r, C, S, \underline{\mathbf{P}}_{K_S^{-}}(?n)). \\
 & & \quad\quad\quad r : \mathbf{H}(n_C, N_S, n) \blacktriangleright ((r, \vec{p}).P)
\end{array}
$$

Table 6. Bilateral *TLS* protocol for the service between C and S, where $C' = (C, K_C^{\pm})$, $S' = (S, K_S^{\pm})$ and $CA = K_{CA}^{\pm}$.

$$
\begin{array}{lcl}
\overline{r}[\mathit{TLS}_2, C', S', \mathit{CA}] \rhd \langle r, \vec{v}\rangle.P & \triangleq & (\nu\ N_C)\langle r, C, S, N_C\rangle. \\
 & & \quad (r, S, C, ?n_S, \underline{\mathbf{S}}_{K_{CA}^{+}}(S, ?x_{ks})). \\
 & & \quad (\nu\ N) \\
 & & \quad\ \langle r, C, S, \mathbf{S}_{K_{CA}^{-}}(C, K_C^{+}), \mathbf{P}_{x_{ks}}(N), \mathbf{S}_{K_C^{-}}(N_C, n_S, N)\rangle. \\
 & & \quad\ r : \mathbf{H}(N_C, n_S, N) \blacktriangleright \langle r, \vec{v}\rangle.P \\
r[\mathit{TLS}_2, C', S', \mathit{CA}] \rhd (r, \vec{p}).P & \triangleq & (r, C, S, ?n_C). \\
 & & \quad (\nu\ N_S)\langle r, S, C, N_S, \mathbf{S}_{K_{CA}^{-}}(S, K_S^{+})\rangle. \\
 & & \quad (r, C, S, \underline{\mathbf{S}}_{K_{CA}^{+}}(C, ?kc), \underline{\mathbf{P}}_{K_S^{-}}(?n), \underline{\mathbf{S}}_{kc}(n_C, N_S, n)). \\
 & & \quad r : \mathbf{H}(n_C, N_S, n) \blacktriangleright (r, \vec{p}).P
\end{array}
$$

Example 4. Returning to Example 2.2 we can now complete the description of how to make use of the *TLS* protocols. Applying the expansion rules to the system defined in Example 2.2, where the *TLS* protocol has been inlined into the notation, will result in an overall system taking the form

$$\mathit{System} \triangleq (\nu_{\pm}\ K_{CA})(\nu_{\pm}\ K_{Bank})\ (\mathit{Client} \mid \mathit{Bank})$$

where in the case of the unilateral *TLS* protocols the client and bank are now given as follows:

$$
\begin{array}{l}
\mathit{Client} \triangleq \\
\quad (\nu\ TS)(\nu\ r)\langle r_{env}, req, r\rangle.(\nu\ N_C)\langle r, \mathit{Client}, \mathit{Bank}, N_C\rangle. \\
\quad\ (r, \mathit{Bank}, \mathit{Client}, ?n_b, \underline{\mathbf{S}}_{K_{CA}^{+}}(\mathit{Bank}, ?x_{K_{Bank}})) \\
\quad\ (\nu\ N)\langle r, \mathit{Client}, \mathit{Bank}, \mathbf{P}_{x_{K_{Bank}}}(N)\rangle. \\
\quad\quad r : \mathbf{H}(N_C, n_b, N) \blacktriangleright (\ \ \langle r, \mathit{Client}, \mathit{Bank}, TS\rangle. \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ (r, \mathit{Bank}, \mathit{Client}, TS, ?bal).0\ )
\end{array}
$$

$$
\begin{array}{l}
\mathit{Bank} \triangleq \\
\quad (r_{env}, req, ?r).(r, \mathit{Client}, \mathit{Bank}, ?n_c).(\nu\ N_B) \\
\quad \langle r, \mathit{Bank}, \mathit{Client}, N_B, \mathbf{S}_{K_{CA}^{-}}(\mathit{Bank}, K_{Bank}^{+})\rangle \\
\quad (r, \mathit{Client}, \mathit{Bank}, \underline{\mathbf{P}}_{K_{Bank}^{-}}(?n)). \\
\quad r : \mathbf{H}(n_c, N_B, n) \blacktriangleright (\ \ (r, \mathit{Client}, \mathit{Bank}, ?ts). \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\ (\nu\ Bal)\ \langle r, \mathit{Bank}, \mathit{Client}, ts, Bal\rangle.0\ )
\end{array}
$$

The concrete specification is close to the style of process algebras with cryptography, e.g. the spi-calculus [AG99] and the LySa calculus [BBD^{+}05], where there are no constructs for services. □

Table 7. The CaPiTo concrete level structural congruence.

$P_1 \mid (\nu\ m)P_2 \equiv (\nu\ m)(P_1 \mid P_2)$	if $m \notin \mathsf{fn}(P_1)$	
$P_1 \mid (\nu_\pm\ m)P_2 \equiv (\nu_\pm\ m)(P_1 \mid P_2)$	if $\{m^+, m^-\} \cap \mathsf{fn}(P_1) = \emptyset$	
$r : e \blacktriangleright (\nu\ m)P \equiv (\nu\ m)(r : e \blacktriangleright P)$	if $m \notin \mathsf{fn}(r, e)$	
$r : e \blacktriangleright (\nu_\pm\ m)P \equiv (\nu_\pm\ m)(r : e \blacktriangleright P)$	if $\{m^+, m^-\} \cap \mathsf{fn}(r, e) = \emptyset$	
$P_1 \equiv P_2 \quad \text{if } P_1 \triangleq P_2$		$r : e \blacktriangleright 0 \equiv 0$
$(\nu\ m)(\nu\ n)P \equiv (\nu\ n)(\nu\ m)P$		$(\nu\ m)0 \equiv 0$
$(\nu_\pm\ m)(\nu_\pm\ n)P \equiv (\nu_\pm\ n)(\nu_\pm\ m)P$		$(\nu_\pm\ m)0 \equiv 0$
$!P \equiv P \mid !P$		$P_1 \mid P_2 \equiv P_2 \mid P_1$
$P \mid 0 \equiv P$		$(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$

Table 8. The CaPiTo concrete level labelled transition system.

(out)	$\langle r, \vec{v}\rangle.P \xrightarrow{\langle r, \vec{v}\rangle} P$	(in)	$\dfrac{\mathcal{M}(\vec{v}, \vec{p}) = \sigma}{(r, \vec{p}).P \xrightarrow{(r, \vec{v})} P\sigma}$
(t-out)	$\dfrac{P \xrightarrow{\langle r, \vec{v}\rangle} P'}{r : w \blacktriangleright P \xrightarrow{\langle r, E_w\langle \vec{v}\rangle\rangle} r : w \blacktriangleright P'}$	(t-in)	$\dfrac{P \xrightarrow{(r, \vec{v})} P'}{r : w \blacktriangleright P \xrightarrow{(r, E_w(\vec{v}))} r : w \blacktriangleright P'}$
(sync)	$\dfrac{P_1 \xrightarrow{\langle r, \vec{v}\rangle} P_1' \quad P_2 \xrightarrow{(r, \vec{v})} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$	(t-sync)	$\dfrac{P_1 \xrightarrow{\langle r, E_w\langle \vec{v}\rangle\rangle} P_1' \quad P_2 \xrightarrow{(r, E_w(\vec{v}))} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$
(r-pass1)	$\dfrac{P \xrightarrow{\lambda} P'}{(\nu\ n)P \xrightarrow{\lambda} (\nu\ n)P'}$ if $n \notin \mathsf{n}(\lambda)$	(r-pass2)	$\dfrac{P \xrightarrow{\lambda} P'}{(\nu_\pm\ n)P \xrightarrow{\lambda} (\nu_\pm\ n)P'}$ if $\{n^+, n^-\} \cap \mathsf{n}(\lambda) = \emptyset$
(t-pass)	$\dfrac{P \xrightarrow{\lambda} P'}{r : w \blacktriangleright P \xrightarrow{\lambda} r : w \blacktriangleright P'}$ $\lambda \ = \ \langle r', \vec{v}\rangle \mid (r', \vec{p})$ $\mid \ \langle r', E_w\langle \vec{v}\rangle\rangle \mid (r', E_w(\vec{v})) \mid \tau$ and $r \neq r'$	(par)	$\dfrac{P \xrightarrow{\lambda} P'}{P \mid P_1 \xrightarrow{\lambda} P' \mid P_1}$ if $\mathsf{fn}(P_1) \cap \mathsf{bn}(\lambda) = \emptyset$
(congr)	$\dfrac{P \equiv P' \quad P' \xrightarrow{\lambda} P'' \quad P'' \equiv P'''}{P \xrightarrow{\lambda} P'''}$	(choice)	$\dfrac{P_1 \xrightarrow{\lambda} P_1'}{P_1 + P_2 \xrightarrow{\lambda} P_1'}$

Semantics of the concrete level. The concrete level semantics consists of a structural congruence and a labelled transition system.

The structural congruence $\equiv$ is defined as the least congruence relation induced by the laws in Table 7. The first rules are known as the "scope extension" rules. They describe how a bound name m may be extruded by an output action, causing the scope of m to be extended; here $\mathsf{fn}()$ denotes the set of free names. The rule $P_1 \equiv P_2$ if $P_1 \triangleq P_2$ is more general than the usual rule $A \equiv P$ if $A \triangleq P$ for unfolding recursive definitions; this is essential to the CaPiTo approach of unfolding protocol stacks thereby passing from the plug-ing level to the concrete level. We shall provide several examples of this in Section 3.

The labelled transition relation $\xrightarrow{\lambda}$ is induced by the rules in Table 8. The label λ is generated by the following grammar:

$$\lambda ::= \tau \mid (r, \vec{v}) \mid \langle r, \vec{v}\rangle \mid \langle r, \mathbf{E}_w\langle \vec{v}\rangle\rangle \mid (r, \mathbf{E}_w(\vec{v}))$$

Here the labels $\langle r, \vec{v}\rangle$ and $(r, \vec{v})$ result from applying rules for output (*out*) and input (*in*), respectively; $\mathbf{E}_w\langle \vec{v}\rangle$ and $\mathbf{E}_w(\vec{v})$ indicate encryption and decryption using the key w; finally, we use τ to denote a silent transaction. Names, $\mathsf{n}(\lambda)$, bound names, $\mathsf{bn}(\lambda)$, and free names, $\mathsf{fn}(\lambda)$, are defined in the usual way [Mil99]. We explain the rules below.

Rules (out) and (in) describe how a list of values $\vec{v}$ is output and how it is then input and matched to a pattern

$\vec{p}$ creating new variable bindings recorded in the substitution σ that is then applied to the continuation of the input operation.

The pattern matching function, $\mathcal{M}$, is defined as follows (or see Appendix A for a more verbose definition). The substitution $\mathcal{M}(\vec{v}, \vec{p}) = \sigma$ resulting from matching $\vec{v}$ against $\vec{p}$ is intended to ensure that $\vec{v}$ and $\vec{p}\sigma$ only differ in their use of asymmetric keys: a public key in one should relate to the corresponding private key in the other. If we write $switch(K_A^+) = K_A^-$ and $switch(K_A^-) = K_A^+$ we may define $\mathcal{M}$ as follows

$$\mathcal{M}(\vec{v}, \vec{p}) \quad = \quad \begin{cases} \sigma & \text{if } switch(\vec{p}\sigma) = \vec{v} \text{ and } dom(\sigma) = \mathsf{dv}(\vec{p}) \\ undef & \text{otherwise} \end{cases}$$

where $\mathsf{dv}()$ denotes the set of defined variables (those with a question mark in front of them) and where we use a notion of substitution where $v\sigma = v$, $(p_1, \ldots, p_k)\sigma = (p_1\sigma, \ldots, p_k\sigma)$, $(?x)\sigma = \sigma(x)$, and $x\sigma =$ undefined.

Returning to Table 8, the rules (t-out) and (t-in) model communications inside a tunnel protected by the symmetric key w. Rule (t-sync) describes the communication inside the tunnel; finally, rule (t-pass) propagates all the activities that are transparent to tunnels. Rules (choice), (par), (r-pass1) and (r-pass2) are the standard ones for nondeterministic choice, parallel composition and restriction. and structural congruence. Rule (congr) is for the structural congruence. It makes sure the processes before and after the transition, P and P''', are in the right forms.

3. Service-Oriented Examples

To illustrate the use of the CaPiTo approach to specification of service-oriented systems we shall now look at the financial Credit Request case study developed in the context of the EU project SENSORIA [Sen] and the SAML single sign-on protocol [Lib].

3.1. The Credit Request Case Study

Abstract specification. Here a client C requests a credit from the validation service VS of a bank. Once the bank has obtained the request it will invoke a service at one of two specialised departments, one taking care of larger enterprises Ser_E and one taking care of smaller companies Ser_C. The overall system is specified as follows at the abstract CaPiTo level; we shall comment on it below:

$$\begin{array}{rcl} C & \triangleq & !\ (\nu\ Bta)\overline{req\nu}[\ ].(\langle Bta\rangle.(Bta, ?x_r).\uparrow\langle x_r\rangle.0) \\ VS & \triangleq & !\ req\nu[\ ].((?y_{bta}).(\\ & & \quad \overline{val_E\nu}[\ ].(\langle y_{bta}\rangle.(?y_r).\uparrow\langle y_{bta}, y_r\rangle.0) \\ & & +\ \overline{val_C\nu}[\ ](\langle y_{bta}\rangle.(?y_r).\uparrow\langle y_{bta}, y_r\rangle.0)) \\ Ser_E & \triangleq & !\ val_E\nu[\ ].((?z_{bta}).\langle isValid(z_{bta})\rangle.0) \\ Ser_C & \triangleq & !\ val_C\nu[\ ].((?z_{bta}).\langle isValid(z_{bta})\rangle.0) \end{array}$$

First the client C invokes the service req at the bank by sending its Balance Total Assets (Bta). The validation service VS will handle the request by invoking a validation service val either at Ser_E or at Ser_C depending on the status of the client, modelled as a non-deterministic choice between two almost identical sub-processes of the form $\overline{val\nu}[\ ](\langle y_{bta}\rangle.(?y_r).\uparrow\langle y_{bta}, y_r\rangle.0)$. The invocation forwards the balance Bta obtained from the client and the response (recorded in the variable y_r) will tell whether or not the enquiry was valid. Having obtained this answer the validation service VS will send a message back to the client using the construct $\uparrow\langle y_{bta}, y_r\rangle$. The whole system is thus obtained as the parallel composition of the four processes:

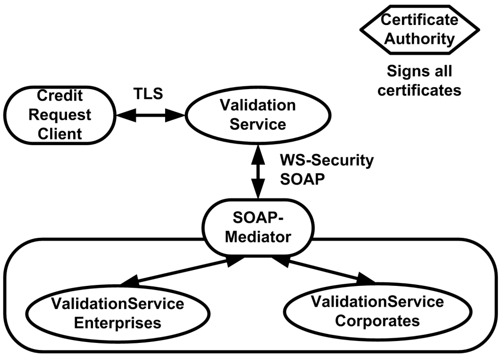

Fig. 1. The Credit Request Case Study.

$$System \triangleq C \mid VS \mid Ser_C \mid Ser_E$$

Plug-in specification. The case study goes one step further and gives details about the various protocols needed to secure the communication [NAPN] as shown in Figure 1. The service *req* provided by the validation service *VC* and invoked by the client *C* should be protected by the *TLS* protocol. The validation service delegates the validation of the balance to the correct service via WS-Security [WS-] and make use of a *SOAP*-Mediator (*SM*) [SOA], that works as an application level router (using WS-Addressing), and is responsible for invoking the service *val* offered by the two specialised departments.

Using the CaPiTo approach we can get a much more modular specification of the scenario than the one given in [NAPN] where all of this is mixed together in a single narration. The idea is simply to specify the relevant plug-ins and then extend the abstract specification with the required information

$$\begin{aligned}
C &\triangleq\ !\ (\nu\ Bta)\overline{req\nu}[TLS, C, S, CA].(\langle Bta\rangle.\\
&\qquad (Bta, ?x_r).\uparrow\langle x_r\rangle.0)\\
VS &\triangleq\ !\ req\nu[TLS, C, S, CA].((?y_{bta}).(\\
&\qquad \overline{val_E\nu}[P2P^+, S, R_E, CA; SOAP, VS, SM, Ser_E].\\
&\qquad (\langle y_{bta}\rangle.(?y_r).\uparrow\langle y_{bta}, y_r\rangle.0)\\
&\qquad + \overline{val_C\nu}[P2P^+, S, R_C, CA; SOAP, VS, SM, Ser_C]\\
&\qquad (\langle y_{bta}\rangle.(?y_r).\uparrow\langle y_{bta}, y_r\rangle.0))\\
Ser_E &\triangleq\ !\ val_E\nu[P2P^+, S, R_E, CA; SOAP, VS, SM, Ser_E].\\
&\qquad ((?z_{bta}).\langle isValid(z_{bta})\rangle.0)\\
Ser_C &\triangleq\ !\ val_C\nu[P2P^+, S, R_C, CA; SOAP, VS, SM, Ser_C].\\
&\qquad ((?z_{bta}).\langle isValid(z_{bta})\rangle.0)
\end{aligned}$$

where $CA = K^{\pm}_{CA}$, $S = (VS, K^{\pm}_{VS})$, $R_E = (Ser_E, K^{\pm}_{Ser_E})$ and $R_C = (Ser_C, K^{\pm}_{Ser_C})$.

The protocol *SOAP* (Simple Object Access Protocol) [SOA] is used to exchange data in a decentralised distributed scenario and only defines the message format. In this case study, *SOAP* works by incorporating a few additional fields (i.e. sender, receiver) in messages.

The *TLS* protocol is already presented in Table 5 so let us have a look at the communications between the validation service *VS* and the services Ser_E and Ser_C. As already mentioned they are to be protected by Web-Service Security (WS-Security) and to be routed by a *SOAP* mediator.

WS-Security [WS-] is a communication protocol suite providing security to Web Services, while guaranteeing end-to-end integrity, authenticity and privacy. Furthermore, WS-Security gives a standardised format for the

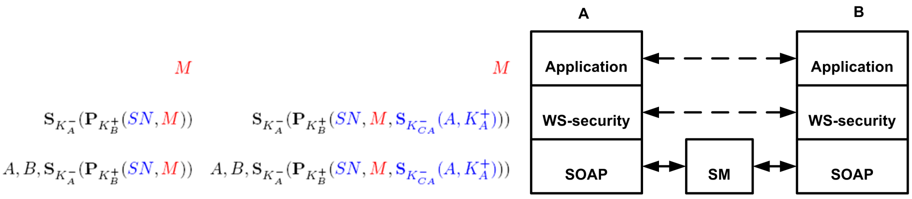
A
B
M
M
Application
Application
$\mathbf{S}_{K_A^-}(\mathbf{P}_{K_B^+}(SN, M))$
$\mathbf{S}_{K_A^-}(\mathbf{P}_{K_B^+}(SN, M, \mathbf{S}_{K_{CA}^-}(A, K_A^+)))$
WS-security
WS-security
$A, B, \mathbf{S}_{K_A^-}(\mathbf{P}_{K_B^+}(SN, M))$
$A, B, \mathbf{S}_{K_A^-}(\mathbf{P}_{K_B^+}(SN, M, \mathbf{S}_{K_{CA}^-}(A, K_A^+)))$
SOAP
SM
SOAP

Fig. 2. Protocol Stack of the Credit Request Case Study.

respective *SOAP* messages involved in the protocols. In this case study its use is restricted to signing and encrypting message content, while leaving the message header as plain text, so as to allow *SOAP*-routing.

The protocol stack used in the case study therefore contains WS-Security and *SOAP*. The structure of the stack and the operation of WS-Security (both the variants *P2P* and $P2P^+$) and *SOAP* are illustrated in Fig. 2.

These protocols are modelled as plug-ins to the service requests and responses. The whole system is defined as

$$\begin{aligned} \mathit{System} \triangleq\ & (\nu_\pm\ K_{VS})(\nu_\pm\ K_{Ser_E})(\nu_\pm\ K_{Ser_C}) \\ & \quad C \mid \mathit{VS} \mid \mathit{SM} \mid \mathit{Ser}_E \mid \mathit{Ser}_C \end{aligned}$$

where the *SOAP* mediator *SM* is given by:

$$\mathit{SM} \triangleq !\ (?r, ?A, \mathit{SM}, A, ?B, ?M).\langle r, \mathit{SM}, B, A, B, M\rangle$$

and where we assume that the public key of the Certificate Authority CA, K_{CA}^+, is known to all parties involved.

Example 5. Applying the transformation function $\mathcal{T}$ on *VS*'s plug-in specification gives the following result (except that for simplicity, only one branch of *VS* is considered here):

$$\begin{aligned} & \mathcal{T}(\mathit{VS}, [r_{env}]) \triangleq \\ & \quad (r_{env}, req, ?r_1).r_1[\mathit{TLS}, C, S, CA] \rhd (\\ & \quad\quad (r_1, ?y_{bta}). \\ & \quad\quad\quad (\nu\ r_2)\langle r_{env}, val_E, r_2\rangle. \\ & \quad\quad\quad\quad \overline{r_2}[\mathit{SOAP}, \mathit{VS}, \mathit{SM}, \mathit{Ser}_E] \rhd (\overline{r_2}[P2P^+, \mathit{VS}, \mathit{Ser}_E, CA] \rhd (\\ & \quad\quad\quad\quad\quad \langle r_2, y_{bta}\rangle.(r_2, ?y_r).\langle r_1, y_{bta}, y_r\rangle.0))) \end{aligned}$$

Concrete specification. The protocols are defined in Table 9 and Table 10. Protocols $P2P^+, P2P$ and *SOAP* are all defined in an inductive way such that each protocol is able to deal with a sequence of messages. The protocol $P2P^+$ deals with outgoing messages (e.g. $\langle r, \vec{v}\rangle.P$) in the following way: it firstly generates a sequence number *SN* for correlating relevant messages, then it encrypts and signs the sequence number *SN*, the message *M* and the sender's certificate, using the receiver's public key, and finally it invokes *P2P* to handle the next message. Reverse actions are taken for incoming messages (e.g. $(r, \vec{p}).P$). Similarly for *P2P* except that here no sender's certificate is included. The definition of *SOAP* is parameterised on three principals, *S* for sender, *SM* for mediator, and *R* for receiver. It includes a few additional fields to messages for specifying the intended sender and receiver.

Table 9. Point-to-point Protocol for the service between A and B, where $S = (A, K^{\pm}_{A})$, $R = (B, K^{\pm}_{B})$, and $CA = K^{\pm}_{CA}$.

$\overline{r}[P2P^{+}, S, R, CA] \rhd (\langle r, \vec{e}\rangle.P)$	$\triangleq$	$(\nu\ SN)\langle r, \mathbf{S}_{K^{-}_{A}}(\mathbf{P}_{K^{+}_{B}}(SN, \vec{e}, \mathbf{S}_{K^{-}_{CA}}(A, K^{+}_{A})))\rangle.$ $\overline{r}[P2P, S, R, SN] \rhd P$
$\overline{r}[P2P^{+}, S, R, CA] \rhd (\langle r', \vec{e}\rangle.P)$	$\triangleq$	$\langle r', \vec{e}\rangle.\overline{r}[P2P^{+}, S, R, CA] \rhd P$ if $r \neq r'$
$\overline{r}[P2P^{+}, S, R, CA] \rhd ((\nu\ n)P)$	$\triangleq$	$(\nu\ n)\overline{r}[P2P^{+}, S, R, CA] \rhd P$
$\overline{r}[P2P^{+}, S, R, CA] \rhd (P_1 + P_2)$	$\triangleq$	$(\overline{r}[P2P^{+}, S, R, CA] \rhd P_1)+$ $(\overline{r}[P2P^{+}, S, R, CA] \rhd P_2)$
$\overline{r}[P2P^{+}, S, R, CA] \rhd (P_1 \mid P_2)$	$\triangleq$	$(\overline{r}[P2P^{+}, S, R, CA] \rhd P_1) \mid (\overline{r}[P2P^{+}, S, R, CA] \rhd P_2)$
$\overline{r}[P2P^{+}, S, R, CA] \rhd (!P)$	$\triangleq$	$!(\overline{r}[P2P^{+}, S, R, CA] \rhd P)$
$\overline{r}[P2P^{+}, S, R, CA] \rhd 0$	$\triangleq$	0
$r[P2P^{+}, S, R, CA] \rhd ((r, \vec{p}).P)$	$\triangleq$	$(r, \underline{\mathbf{S}}_{K^{+}_{A}}(\underline{\mathbf{P}}_{K^{-}_{B}}(?sn, \vec{p}, \underline{\mathbf{S}}_{K^{+}_{CA}}(A, ?k^{+}_{a})))).$ $r[P2P, (A, k^{+}_{a}), R, sn] \rhd P$
$r[P2P^{+}, S, R, CA] \rhd ((r', \vec{p}).P)$	$\triangleq$	$(r', \vec{p}).r[P2P^{+}, S, R, CA] \rhd P$ if $r \neq r'$
$r[P2P^{+}, S, R, CA] \rhd ((\nu\ n)P)$	$\triangleq$	$(\nu\ n)r[P2P^{+}, S, R, CA] \rhd P$
$r[P2P^{+}, S, R, CA] \rhd (P_1 + P_2)$	$\triangleq$	$(r[P2P^{+}, S, R, CA] \rhd P_1)+$ $(r[P2P^{+}, S, R, CA] \rhd P_2)$
$r[P2P^{+}, S, R, CA] \rhd (P_1 \mid P_2)$	$\triangleq$	$(r[P2P^{+}, S, R, CA] \rhd P_1) \mid (r[P2P^{+}, S, R, CA] \rhd P_2)$
$r[P2P^{+}, S, R, CA] \rhd (!P)$	$\triangleq$	$!(r[P2P^{+}, S, R, CA] \rhd P)$
$r[P2P^{+}, S, R, CA] \rhd 0$	$\triangleq$	0
$\overline{r}[P2P, L, M, SN] \rhd (\langle r, \vec{e}\rangle.P)$	$\triangleq$	$\langle r, \mathbf{S}_{K^{-}_{A}}(\mathbf{P}_{K^{+}_{B}}(SN, \vec{e}))\rangle.\overline{r}[P2P, L, M, SN] \rhd P$
$\overline{r}[P2P, L, M, SN] \rhd (\langle r', \vec{e}\rangle.P)$	$\triangleq$	$\langle r', \vec{e}\rangle.\overline{r}[P2P, L, M, SN] \rhd P$ if $r \neq r'$
$\overline{r}[P2P, L, M, SN] \rhd ((r, \vec{p}).P)$	$\triangleq$	$(r, \underline{\mathbf{S}}_{K^{+}_{A}}(\underline{\mathbf{P}}_{K^{-}_{B}}(SN, \vec{p}))).\overline{r}[P2P, L, M, SN] \rhd .P$
$\overline{r}[P2P, L, M, SN] \rhd ((r', \vec{p}).P)$	$\triangleq$	$(r', \vec{p}).\overline{r}[P2P, L, M, SN] \rhd P$ if $r \neq r'$
$\overline{r}[P2P, L, M, SN] \rhd ((\nu\ n)P)$	$\triangleq$	$(\nu\ n)\overline{r}[P2P, L, M, SN] \rhd P$
$\overline{r}[P2P, L, M, SN] \rhd (P_1 + P_2)$	$\triangleq$	$(\overline{r}[P2P, L, M, SN] \rhd P_1) + (\overline{r}[P2P, L, M, SN] \rhd P_2)$
$\overline{r}[P2P, L, M, SN] \rhd (P_1 \mid P_2)$	$\triangleq$	$(\overline{r}[P2P, L, M, SN] \rhd P_1) \mid (\overline{r}[P2P, L, M, SN] \rhd P_2)$
$\overline{r}[P2P, L, M, SN] \rhd (!P)$	$\triangleq$	$!(\overline{r}[P2P, L, M, SN] \rhd P)$
$\overline{r}[P2P, L, M, SN] \rhd 0$	$\triangleq$	0
$r[P2P, L, M, sn] \rhd (\langle r, \vec{e}\rangle.P)$	$\triangleq$	$\langle r, \mathbf{S}_{K^{-}_{A}}(\mathbf{P}_{K^{+}_{B}}(sn, \vec{e}))\rangle.r[P2P, L, M, sn] \rhd P$
$r[P2P, L, M, sn] \rhd (\langle r', \vec{e}\rangle.P)$	$\triangleq$	$\langle r', \vec{e}\rangle.r[P2P, L, M, sn] \rhd P$ if $r \neq r'$
$r[P2P, L, M, sn] \rhd ((r, \vec{p}).P)$	$\triangleq$	$(r, \underline{\mathbf{S}}_{K^{+}_{A}}(\underline{\mathbf{P}}_{K^{-}_{B}}(sn, \vec{p}))).r[P2P, L, M, sn] \rhd P$
$r[P2P, L, M, sn] \rhd ((r', \vec{p}).P)$	$\triangleq$	$(r', \vec{p}).r[P2P, L, M, sn] \rhd P$ if $r \neq r'$
$r[P2P, L, M, sn] \rhd ((\nu\ n)P)$	$\triangleq$	$(\nu\ n)r[P2P, L, M, sn] \rhd P$
$r[P2P, L, M, sn] \rhd (P_1 + P_2)$	$\triangleq$	$(r[P2P, L, M, sn] \rhd P_1) + (r[P2P, L, M, sn] \rhd P_2)$
$r[P2P, L, M, sn] \rhd (P_1 \mid P_2)$	$\triangleq$	$(r[P2P, L, M, sn] \rhd P_1) \mid (r[P2P, L, M, sn] \rhd P_2)$
$r[P2P, L, M, sn] \rhd (!P)$	$\triangleq$	$!(r[P2P, L, M, sn] \rhd P)$
$r[P2P, L, M, sn] \rhd 0$	$\triangleq$	0

Example 6. Applying the unfolding function on the above result gives the concrete level specification of *VS*:

$$
\begin{array}{l}
(r_{env}, req, ?r_1).(r_1, C, VS, ?n_C).\\
\ (\nu\ N_S)\langle r_1, VS, C, N_S, \mathbf{S}_{K^{-}_{CA}}(VS, K^{+}_{VS})\rangle\\
\ \ (r_1, C, VS, \underline{\mathbf{P}}_{K^{-}_{VS}}(?n)).\\
\ \ \ r_1 : \mathbf{H}(n_C, N_S, n) \blacktriangleright (\\
\ \ \ \ (r_1, ?y_{bta}).\\
\ \ \ \ \ (\nu\ r_2)\langle r_{env}, val_E, r_2\rangle.(\nu\ SN)\\
\ \ \ \ \ \ \langle r_2, VS, SM, VS, Ser_E, \mathbf{S}_{K^{-}_{VS}}(\mathbf{P}_{K^{+}_{Ser_E}}(SN, y_{bta}, \mathbf{S}_{K^{-}_{CA}}(VS, K^{+}_{VS}))))\rangle.\\
\ \ \ \ \ \ \ (r_2, SM, VS, Ser_E, VS, \underline{\mathbf{S}}_{K^{+}_{Ser_E}}(\underline{\mathbf{P}}_{K^{-}_{VS}}(SN, ?y_r))).\\
\ \ \ \ \ \ \ \ \langle r_1, y_{bta}, y_r\rangle.0)
\end{array}
$$

Table 10. SOAP Protocol for the service between sender S and responder R using SM as $SOAP$ Mediator.

$$
\begin{array}{lcl}
\overline{r}[SOAP, S, SM, R] \triangleright (\langle r, \vec{e}\rangle.P) & \triangleq & \langle r, S, SM, S, R, \vec{e}\rangle.\overline{r}[SOAP, S, SM, R] \triangleright P \\
\overline{r}[SOAP, S, SM, R] \triangleright (\langle r', \vec{e}\rangle.P) & \triangleq & \langle r', \vec{e}\rangle.\overline{r}[SOAP, S, SM, R] \triangleright P \ \text{ if } r \neq r' \\
\overline{r}[SOAP, S, SM, R] \triangleright ((r, \vec{p}).P) & \triangleq & (r, SM, S, R, S, \vec{p}).\overline{r}[SOAP, S, SM, R] \triangleright P \\
\overline{r}[SOAP, S, SM, R] \triangleright ((r', \vec{p}).P) & \triangleq & (r', \vec{p}).\overline{r}[SOAP, S, SM, R] \triangleright P \ \text{ if } r \neq r' \\
\overline{r}[SOAP, S, SM, R] \triangleright ((\nu\ n)P) & \triangleq & (\nu\ n)\overline{r}[SOAP, S, SM, R] \triangleright P \\
\overline{r}[SOAP, S, SM, R] \triangleright (P_1 + P_2) & \triangleq & (\overline{r}[SOAP, S, SM, R] \triangleright P_1)+ \\
 & & \quad (\overline{r}[SOAP, S, SM, R] \triangleright P_2) \\
\overline{r}[SOAP, S, SM, R] \triangleright (P_1 | P_2) & \triangleq & (\overline{r}[SOAP, S, SM, R] \triangleright P_1)| \\
 & & \quad (\overline{r}[SOAP, S, SM, R] \triangleright P_2) \\
\overline{r}[SOAP, S, SM, R] \triangleright (!P) & \triangleq & !(\overline{r}[SOAP, S, SM, R] \triangleright P) \\
\overline{r}[SOAP, S, SM, R] \triangleright 0 & \triangleq & 0 \\
r[SOAP, S, SM, R] \triangleright (\langle r, \vec{e}\rangle.P) & \triangleq & \langle r, R, SM, R, S, \vec{e}\rangle.r[SOAP, S, SM, R] \triangleright P \\
r[SOAP, S, SM, R] \triangleright (\langle r', \vec{e}\rangle.P) & \triangleq & \langle r', \vec{e}\rangle.r[SOAP, S, SM, R] \triangleright P \ \text{ if } r \neq r' \\
r[SOAP, S, SM, R] \triangleright ((r, \vec{p}).P) & \triangleq & (r, SM, R, S, R, \vec{p}).r[SOAP, S, SM, R] \triangleright P \\
r[SOAP, S, SM, R] \triangleright ((r', \vec{p}).P) & \triangleq & (r', \vec{p}).r[SOAP, S, SM, R] \triangleright P \ \text{ if } r \neq r' \\
r[SOAP, S, SM, R] \triangleright ((\nu\ n)P) & \triangleq & (\nu\ n)r[SOAP, S, SM, R] \triangleright P \\
r[SOAP, S, SM, R] \triangleright (P_1 + P_2) & \triangleq & (r[SOAP, S, SM, R] \triangleright P_1)+ \\
 & & \quad (r[SOAP, S, SM, R] \triangleright P_2) \\
r[SOAP, S, SM, R] \triangleright (P_1 | P_2) & \triangleq & (r[SOAP, S, SM, R] \triangleright P_1)| \\
 & & \quad (r[SOAP, S, SM, R] \triangleright P_2) \\
r[SOAP, S, SM, R] \triangleright (!P) & \triangleq & !(r[SOAP, S, SM, R] \triangleright P) \\
r[SOAP, S, SM, R] \triangleright 0 & \triangleq & 0
\end{array}
$$

3.2. The SAML Single Sign-On Protocol

SAML stands for Security Assertion Markup Language and has been developed by the standardisation organisation OASIS [OAS]. They also developed the Single Sign-On protocol that forms the basis for commercial protocols developed within projects such as the Liberty-Alliance Project [Lib] and the Shibboleth© Project [Shi].

The SAML Single Sign-On protocol (SAML SSO for short) is used to authenticate users. The idea is that users only need to authenticate themselves to a single authentication server rather than to a number of individual services – and once that has happened the user has access to all the services and hence does not need to go through additional logon procedures. The authentication service manages the accounts of the individual users and therefore the individual services do not need to take care of this.

In the SAML SSO scenario we thus distinguish between three kinds of principals:

User: A user U accesses the network via a standard web browser implementing standard protocols such as HTTP and TLS.

Destination: A destination site D offers a restricted service which is only available to users that have been authenticated by a central authentication server.

Server: The server site S will take care of authenticating the users, thereby allowing them to access services at the destination sites.

The protocol uses the concept of an *artifact* that arises because most browsers only allow for a limited length of the HTTP URL messages. Many websites send arguments along with the HTTP query by appending them to the query. For example searching for the string *SSO* on *google* results in the URL `http://www.google.com/search?q=SSO`. In this example the URL is 35 characters long, but in some cases an argument might 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. An artifact is in fact a sequence of bytes and in our model we shall encode it as a nonce. In the SAML scenario, the artifact is used as a pointer pointing to a larger

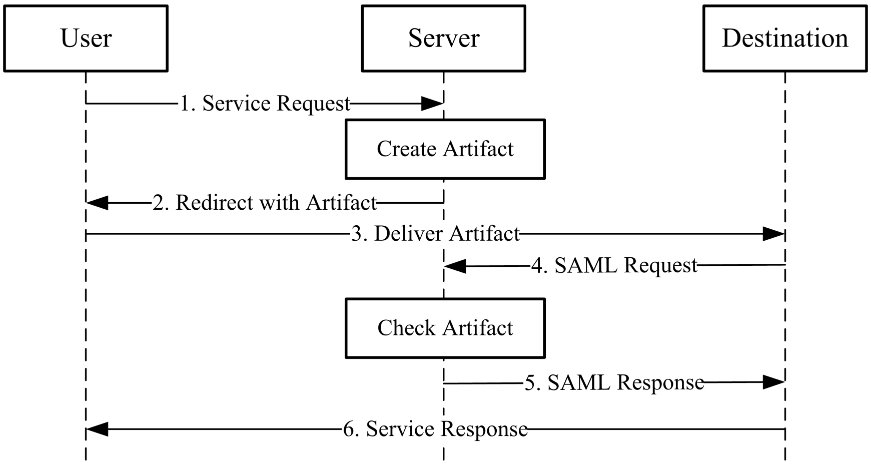

Fig. 3. The SAML SSO Protocol.

SAML assertion, which contains authentication and authorization decision statements that destinations use to make access control decision [HM04].

Prior to usage of the SAML SSO protocol, the user and the server must share a key (K_U); this key is used by the destination sites to communicate confidential data to the user. Based on this, Figure 3 illustrates the six steps of the SAML SSO protocol:

1. The user U makes a request to the authentication server S for a SAML assertion to a specific service at a destination site D by sending the name D.
2. The server S then generates an artifact A and the corresponding SAML assertion, and responds to the user with the artifact that will be used to identify the session.
3. The user sends the artifact to the destination site.
4. To verify the identity of the user the destination now forwards the artifact to the server.
5. The server looks up the SAML assertion using the artifact. If it is verified, the server then sends back a SAML Response message to the destination containing a key K_U that is to be used for communication with the user.
6. The destination and the user communicate with each other using K_U as the key for encryption and decryption.

Abstract specification. Turning to the abstract CaPiTo specification the system is defined as

$$SSO \triangleq (\nu\ K_U)\ (U \mid D \mid S)$$

where

$$\begin{array}{rcl}
U & \triangleq & \overline{inv\nu}[\ ].\langle U, S, D\rangle.(S, U, D, ?a). \\
 & & \quad \overline{fwd\nu}[\ ].\langle U, D, S, a\rangle. \\
 & & \qquad send\nu[\ ].(?m).0 \\
 & & \\
S & \triangleq & inv\nu[\ ].(U, S, ?d)(\nu\ A)\langle S, U, d, A\rangle. \\
 & & \quad val\nu[\ ].(d, S, A).\langle S, d, K_U, U\rangle.0 \\
 & & \\
D & \triangleq & fwd\nu[\ ].(U, D, S, ?a'). \\
 & & \quad \overline{val\nu}[\ ].\langle D, S, a'\rangle.(S, D, ?k_U, U). \\
 & & \qquad (\nu\ M)\overline{send\nu}[\ ]\langle M\rangle.0
\end{array}$$

Here *inv* is a service offered by the authentication server S and it is invoked by the user U. The destination D is offering the *fwd* service and it is invoked by the user upon receipt of the artifact from the server. The destination is then invoking the service *val* provided by the server in order to get information about the encryption service (to be called *enc*) offered by the user.

Plug-in specification. Following [HSN05] and [OAS] we may protect the services using a unilateral (Table 5) or a bilateral (Table 6) TLS protocol. As already mentioned the unilateral version of TLS relies on the server to hold a certificate issued by the Certificate Authority; in the bilateral version the client is assumed to have a similar certificate.

Following the recommendations of the SAML SSO documents we may protect the connections by unilateral TLS protocols. This is expressed by:

$$\begin{array}{rcl}
U & \triangleq & \overline{inv\nu}[TLS, U, S, CA].\langle U, S, D\rangle.(S, U, D, ?a). \\
 & & \quad \overline{fwd\nu}[TLS, U, D, CA].\langle U, D, S, a\rangle. \\
 & & \qquad send\nu[enc, K_U].(?m).0
\end{array}$$

$$\begin{array}{rcl}
S & \triangleq & inv\nu[TLS, U, S, CA].(U, S, ?d)(\nu\ A)\langle S, U, d, A\rangle. \\
 & & \quad val\nu[TLS, D, S, CA].(d, S, A).\langle S, d, K_U, U\rangle.0
\end{array}$$

$$\begin{array}{rcl}
D & \triangleq & fwd\nu[TLS, U, D, CA].(U, D, S, ?a'). \\
 & & \quad \overline{val\nu}[TLS, D, S, CA].\langle D, S, a'\rangle.(S, D, ?k_U, U). \\
 & & \qquad (\nu\ M)\overline{send\nu}[enc, k_U]\langle M\rangle.0
\end{array}$$

Example 7. Applying the transformation function $\mathcal{T}$ on U's plug-in specification gives the following result:

$$\begin{array}{l}
\mathcal{T}(U, [r_{env}]) \triangleq \\
(\nu\ r_1)\langle r_{env}, inv, r_1\rangle.\overline{r_1}[TLS, U, S, CA] \rhd \langle r_1, U, S, D\rangle.(r_1, S, U, D, ?a). \\
\quad (\nu\ r_2)\langle r_{env}, fwd, r_2\rangle.\overline{r_2}[TLS, U, D, CA] \rhd \langle r_2, U, D, S, a\rangle. \\
\qquad (r_{env}, send, ?r_3).r_3[env, K_U] \rhd (r_3, ?m).0
\end{array}$$

Concrete specification. The encryption service of the user can be specified by

$$\begin{array}{rcl}
\overline{r}[enc, key] \rhd (\langle r, \vec{e}\rangle.P) & \triangleq & r : key \blacktriangleright \langle r, \vec{e}\rangle.P \\
r[enc, key] \rhd ((r, \vec{p}).P) & \triangleq & r : key \blacktriangleright (r, \vec{p}).P
\end{array}$$

and simply uses the parameter *key* as a key for encryption and decryption. Thus writing $\overline{send\nu}[enc, key]\langle\vec{e}\rangle.P$ we ensure that $\vec{e}.P$ is encrypted by *key*.

Example 8. Applying the unfolding function on the above result gives the concrete level specification of U.

$$\begin{array}{l}
(\nu\ r_1)\langle r_{env}, inv, r_1\rangle. \\
\ (\nu\ N_U)\langle r_1, U, S, N_U\rangle?(r_1, S, U, ?n_s, \underline{\mathbf{S}}_{K_{CA}^+}(S, ?x_{ks})).(\nu\ N)\langle r_1, U, S, \mathbf{P}_{x_{ks}}(N)\rangle. \\
\ r_1 : \mathbf{H}(N_U, n_s, N) \blacktriangleright (\\
\ \ \langle r_1, U, S, D\rangle.(r_1, S, U, D, ?a). \\
\ \ \ (\nu\ r_2)\langle r_{env}, fwd, r_2\rangle. \\
\ \ \ \ (r_2, D, U, ?n'_s, \underline{\mathbf{S}}_{K_{CA}^+}(D, ?x'_{ks})).(\nu\ N')\langle r_2, U, D, \mathbf{P}_{x'_{ks}}(N')\rangle. \\
\ \ \ \ \ (\nu\ N'_U)\langle r_2, U, D, N'_U\rangle.r_2 : \mathbf{H}(N'_U, n'_s, N') \blacktriangleright (\\
\ \ \ \ \ \ \langle r_2, U, D, S, a\rangle. \\
\ \ \ \ \ \ \ (r_{env}, send, ?r_3). \\
\ \ \ \ \ \ \ \ r_3 : K_U \blacktriangleright (r_3, ?m).0))
\end{array}$$

Clearly it is easy to experiment with different protocols, e.g. bilateral TLS protocol, by simply replacing the plugins, i.e. replacing TLS by TLS$_2$.

Table 11. Transformation of tunnels into LySa.

$$
\begin{array}{l}
r : e_0 \blacktriangleright \langle r', \vec{e}\rangle^{l_0}[dest\ \{\vec{l}\}].P \leadsto \langle r', \{\vec{e}\}^{l_0}_{e_0}[dest\ \{\vec{l}\}]\rangle.r : e_0 \blacktriangleright P \ \ \text{if } r = r' \\
r : e_0 \blacktriangleright \langle r', \vec{e}\rangle^{l_0}[dest\ \{\vec{l}\}].P \leadsto \langle r', \vec{e}\rangle^{l_0}[dest\ \{\vec{l}\}].r : e_0 \blacktriangleright P \ \ \text{if } r \neq r' \\
r : e_0 \blacktriangleright (r', \vec{p})^{l_0}[orig\ \{\vec{l}\}].P \leadsto (r', \{\vec{p}\}^{l_0}_{e_0}[orig\ \{\vec{l}\}]).r : e_0 \blacktriangleright P \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } r = r' \text{ and } \mathsf{fn}(e_0) \cap \cap_1^k \mathsf{fn}(p_i) = \emptyset \\
r : e_0 \blacktriangleright (r', \vec{p})^{l_0}[orig\ \{\vec{l}\}].P \leadsto (r', \vec{p})^{l_0}[orig\ \{\vec{l}\}].r : e_0 \blacktriangleright P \ \ \text{if } r \neq r' \\
r : e_0 \blacktriangleright (\nu\ n)P \leadsto (\nu\ n)r : e_0 \blacktriangleright P \ \ \text{if } n \notin \mathsf{fn}(e_0) \\
r : e_0 \blacktriangleright (\nu_\pm\ n)P \leadsto (\nu_\pm\ n)r : e_0 \blacktriangleright P \ \ \text{if } \{n^+, n^-\} \cap \mathsf{fn}(e_0) = \emptyset \\
r : e_0 \blacktriangleright !P \leadsto !r : e_0 \blacktriangleright P \\
r : e_0 \blacktriangleright P_1 \mid P_2 \leadsto r : e_0 \blacktriangleright P_1 \mid r : e_0 \blacktriangleright P_2 \\
r : e_0 \blacktriangleright P_1 \ + \ P_2 \leadsto r : e_0 \blacktriangleright P_1 \ + \ r : e_0 \blacktriangleright P_2 \\
r : e_0 \blacktriangleright 0 \leadsto 0
\end{array}
$$

4. Static Analysis

We now show how to use the protocol analysis tool LySa [BBD+05] to analyse concrete specifications in CaPiTo. In order for CaPiTo specifications to be compatible with LySa, there are some issues to be addressed.

4.1. From CaPiTo to LySa

Adding Annotations. In order to express our intentions with the protocols, we follow the work of LySa [BBD+05] and manually add annotations about the origin and destination of encrypted messages; this modification is performed in the concrete specification of CaPiTo. The idea is that each encryption occurring in a process is annotated with a crypto-point l defining its position as well as a set of crypto-points $\mathcal{L}$ specifying the destination positions where the encryption is intended to be decrypted. Similarly, each decryption is annotated with a crypto-point defining its position as well as a set of crypto-points specifying the potential origins of encrypted messages to be decrypted. For example, consider the following process:

$$
\begin{array}{ll}
1. & r : e \blacktriangleright \langle r, V_1, V_2\rangle^{l_1}[dest\ \{l_2\}].0\ quad| \\
2. & r : e \blacktriangleright (r, V_1, x)^{l_2}[orig\ \{l_1\}].0
\end{array}
$$

Here the process in line 1 specifies that the encryption is created at crypto-point l_1 and is intended for decryption at crypto-point l_2, whereas the process in line 2 specifies that the message to be decrypted at crypto-point l_2 must come from crypto-point l_1. Similar annotations are made in case of asymmetric encryption, decryption, signature and signature validation.

From Tunnels to LySa. The main difference between concrete specifications in CaPiTo and LySa is the way symmetric encryptions and decryptions are handled. Rather than using tunnels, as discussed in Subsection 2.3, LySa has explicit encryptions (e.g. $\{M\}_K$) and decryptions (e.g. $\mathsf{decrypt}\ V\ \mathsf{as}\ \{;m\}_K$). We therefore transform tunnels into the explicit encryptions and decryptions used to realise them. The transformation rules are listed in Table 11. Note that, in the first four lines, the *dest* and *orig* annotations are passed along so that the resulting forms are compatible with the syntax of LySa. Also replacement in context is implicitly assumed. The transformation is guaranteed to terminate, as each rule inductively unfolds the process until it becomes 0.

4.2. Analysis of LySa

The control flow analysis of LySa [BBD$^+$05] describes a protocol behaviour by collecting all the communications that a process may participate in. In particular, the analysis contains the tuples that may flow over the network and the values that the variables may be bound to. The analysis information is collected in the following main components:

- $\rho : \mathcal{P}(Var \times \mathit{Val})$ is a "global" component containing for each variable the set of names it may be bound to;
- $\kappa : \mathcal{P}(\mathit{Val}^*)$ is a "global" component containing all the tuples that have been communicated;
- $\psi : \mathcal{P}(\mathit{Lab} \times \mathit{Lab})$ is a "local" error component containing an over-approximation of the potential origin/destination violations. If $(l, l') \in \psi$ then something encrypted at crypto-point l might unexpectedly be decrypted at crypto-point l', or something decrypted at l' might have been expected to be encrypted at another place than l.

Formally, the analysis information is represented by a triple (ρ, κ, ψ) called an analysis *estimate* of a given process; for an expression it suffices with a pair (ρ, ϑ) where ϑ is an over-approximation of the set of values to which the expression can evaluate.

Flattening the Pattern Matching. Before going into detail of the analysis rules, we shall introduce an auxiliary function for dealing with pattern matching. We define a judgement for pattern matching, namely $\rho \models_i \vec{p} : \hat{V} \triangleright \hat{W} : \psi$ as shown in Table 12. It is defined in the style of Flow Logic [NNP11] which in particular means that any entities occurring on the righthandsides, and not on the lefthandsides, are implicitly existentially quantified. It traverses the candidate tuple space first in a forward direction, where the tuples in $\hat{V}$ are tested and only tuples satisfying the requirements are carried forward, and then in a backward direction, where the tuples in $\hat{W}$ are those that passed all the requirements. As part of the backward traversal, the judgement also collects *orig*, *dest* annotation violations and includes them in the error component ψ as needed. The number i indicates the next position to be matched.

As an example, the judgement

$$\rho \models_1 (r, A, ?m) : \{(r, A, M), (r, B, M)\} \triangleright \{(r, A, M)\} : \emptyset$$

enforces that m becomes bound to M, and the judgement

$$\begin{aligned}\rho \models_1 \{A, ?m\}_K^{l_1}[\mathit{orig}\ \{l_2\}] : \{\{A, M\}_K^{l_3}[\mathit{dest}\ \{l_4\}], (B, M)\} \triangleright \\ \{\{A, M\}_K\} : \{(l_1, l_3)\}\end{aligned}$$

enforces that the annotation violation (l_3, l_1) is contained in ψ (because $l_1 \notin \{l_4\}$ and $l_3 \notin \{l_2\}$).

Expressions. The judgement for analysing expressions takes the form $\rho \models e : \vartheta$. Basically, the clauses defining the judgement ensure that ϑ contains all the values associated with the components of a term, e.g. a name n evaluates to the set ϑ, provided that n belongs to ϑ; similarly for a variable x, provided that ϑ includes the set of values $\rho(x)$ to which x is associated. The analysis rules are defined in the upper part of Table 13.

Processes. The judgement for analysing processes is $\rho, \kappa \models P : \psi$. For each process P, the analysis ensures that the information in ρ correctly captures the variable bindings taking place, that the information in κ correctly captures the communications taking place, and that annotation violations are correctly recorded in ψ.

The analysis rules are defined in the lower part of Table 13. The (Inp) clause makes use of the auxiliary judgement for pattern matching $\rho \models_i \vec{p} : \hat{V} \triangleright \hat{W}$. It requires that the continuation service P is analysed

Table 12. Flattened pattern matching.

$\rho \models_i \epsilon : \hat{V} \triangleright \hat{W} : \psi$	iff	$\{\vec{v} \in \hat{V} \mid \lvert\vec{v}\rvert = i-1\} \subseteq \hat{W}$
$\rho \models_i n, \vec{p} : \hat{V} \triangleright \hat{W} : \psi$	iff	$\{\vec{v} \in \hat{V} \mid \pi_i(\vec{v}) = n\} \subseteq \hat{V}'\wedge$ $\rho \models_{i+1} \vec{p} : \hat{V}' \triangleright \hat{W} : \psi$
$\rho \models_i x, \vec{p} : \hat{V} \triangleright \hat{W} : \psi$	iff	$\{\vec{v} \in \hat{V} \mid \pi_i(\vec{v}) \in \rho(x)\} \subseteq \hat{V}'\wedge$ $\rho \models_{i+1} \vec{p} : \hat{V}' \triangleright \hat{W} : \psi$
$\rho \models_i ?x, \vec{p} : \hat{V} \triangleright \hat{W} : \psi$	iff	$\rho \models_{i+1} \vec{p} : \hat{V} \triangleright \hat{W} : \psi\wedge$ $\forall \vec{v} : \vec{v} \in \hat{W} \Rightarrow (\pi_i(\vec{v}) \in \rho(x))$
$\rho \models_i \{p_1, \ldots, p_k\}_n^l[orig\ \mathcal{L}], \vec{p} : \hat{V} \triangleright \hat{W} : \psi$	iff	$\{v_1, \ldots, v_k \mid \{v_1, \ldots, v_k\}_n^{l'}[dest\ \mathcal{L}'] \in \pi_i(\hat{V})\} \subseteq \hat{V}'\wedge$ $\rho \models_1 p_1, \ldots, p_k : \hat{V}' \triangleright \hat{W}' : \psi\wedge$ $\{\vec{w} \in \hat{V} \mid \exists \vec{v} \in \hat{W}' : \pi_i(\vec{w}) = \{\vec{v}\}_n\} \subseteq \hat{V}''\wedge$ $\rho \models_{i+1} \vec{p} : \hat{V}'' \triangleright \hat{W} : \psi\wedge$ $(l \notin \mathcal{L}' \vee l' \notin \mathcal{L}) \Rightarrow (l, l') \in \psi$
$\rho \models_i \underline{\mathbf{P}}_{n^-}(p_1, \ldots, p_k)^l[orig\ \mathcal{L}], \vec{p} : \hat{V} \triangleright \hat{W} : \psi$	iff	$\{v_1, \ldots, v_k \mid \mathbf{P}_{n^+}(v_1, \ldots, v_k)^{l'}[dest\ \mathcal{L}'] \in \pi_i(\hat{V})\} \subseteq \hat{V}'\wedge$ $\rho \models_1 p_1, \ldots, p_k : \hat{V}' \triangleright \hat{W}' : \psi\wedge$ $\{\vec{w} \in \hat{V} \mid \exists \vec{v} \in \hat{W}' : \pi_i(\vec{w}) = \mathbf{P}_{n^+}(\vec{v})\} \subseteq \hat{V}''\wedge$ $\rho \models_{i+1} \vec{p} : \hat{V}'' \triangleright \hat{W} : \psi\wedge$ $(l \notin \mathcal{L}' \vee l' \notin \mathcal{L}) \Rightarrow (l, l') \in \psi$
$\rho \models_i \underline{\mathbf{S}}_{n^+}(p_1, \ldots, p_k)^l[orig\ \mathcal{L}], \vec{p} : \hat{V} \triangleright \hat{W} : \psi$	iff	$\{v_1, \ldots, v_k \mid \mathbf{S}_{n^-}(v_1, \ldots, v_k)^{l'}[dest\ \mathcal{L}'] \in \pi_i(\hat{V})\} \subseteq \hat{V}' : \psi\wedge$ $\rho \models_1 p_1, \ldots, p_k : \hat{V}' \triangleright \hat{W}' : \psi\wedge$ $\{\vec{w} \in \hat{V} \mid \exists \vec{v} \in \hat{W}' : \pi_i(\vec{w}) = \mathbf{S}_{n^-}(\vec{v})\} \subseteq \hat{V}''\wedge$ $\rho \models_{i+1} \vec{p} : \hat{V}'' \triangleright \hat{W} : \psi$ $(l \notin \mathcal{L}' \vee l' \notin \mathcal{L}) \Rightarrow (l, l') \in \psi$

only when pattern matching returns a non-empty result. The (Out) clause evaluates all the expressions, $e_1, \ldots, e_k$, and requires that all the combinations of these values are contained in κ. Indeed these are the values that may be communicated. Finally, the continuation service P must be analysed. The rest of the rules are straightforward.

Modelling the attacker. Protocols are executed in an environment where there may exist malicious attackers. For most process algebras, this is modelled as $P_{sys}|Q$ with P_{sys} being the implementation of the protocol and Q representing the actual environment, and this is the scenario we consider as well.

Following [BBD$^+$05], we shall say that a process P is of type $(\mathcal{N}_f, \mathcal{A}_\kappa, \mathcal{A}_{Enc})$ whenever: (1) it is closed (no free variables), (2) its free names are in $\mathcal{N}_f$, (3) all the arities used for sending or receiving are in $\mathcal{A}_\kappa$ and (4) all the arities used for encryption or decryption are in $\mathcal{A}_{Enc}$. Clearly one can inspect P_{sys} to find minimal $\mathcal{N}_f, \mathcal{A}_\kappa, \mathcal{A}_{Enc}$ such that P_{sys} is of type $(\mathcal{N}_f, \mathcal{A}_\kappa, \mathcal{A}_{Enc})$. We also postulate a new name $n_\bullet$, a new variable $x_\bullet$, and a new crypto-point $l_\bullet$ that are not occurring in P_{sys}. We then define the Dolev-Yao attacker's ability as the conjunction of the five components in Table 14 (similar components may be added to deal with asymmetric cryptography).

Implementation outline. The overall goal of the implementation of the analysis is to compute the analysis, e.g. ρ and κ, for a given service P. This is done in two phases. In the first phase, a generation function using Standard ML is constructed, which translates a service into a logic formula in the form of Alternation-free Least Fixed Point logic [NNS02], and in the second phase, a logic solver (the *Succinct solver* [NNS02]) is employed to compute the least interpretations of predicates that satisfy the formula [Gao08].

Table 13. Analysis judgements for expressions and processes.

(Name)	$\rho \models n : \vartheta$	iff	$n \in \vartheta$
(Pri)	$\rho \models n^- : \vartheta$	iff	$n^- \in \vartheta$
(Pub)	$\rho \models n^+ : \vartheta$	iff	$n^+ \in \vartheta$
(Var)	$\rho \models x : \vartheta$	iff	$\rho(x) \subseteq \vartheta$
(Enc)	$\rho \models \{v_1, \ldots, v_k\}^l_{v_0}[dest\ \mathcal{L}] : \vartheta$	iff	$\wedge^k_{i=0} \rho \models v_i : \vartheta_i \wedge$ $\forall w_0, w_1, \ldots, w_k : \wedge^k_{i=0} w_i \in \vartheta_i \Rightarrow$ $\{w_1, \ldots, w_k\}^l_{w_0}[dest\ \mathcal{L}] \in \vartheta$
(AEnc)	$\rho \models \mathbf{P}_{v_0^+}(v_1, \ldots, v_k)^l[dest\ \mathcal{L}] : \vartheta$	iff	$\wedge^k_{i=0} \rho \models v_i : \vartheta_i \wedge$ $\forall w_0, w_1, \ldots, w_k : \wedge^k_{i=0} w_i \in \vartheta_i \Rightarrow$ $\mathbf{P}_{w_0^+}(w_1, \ldots, w_k)^l[dest\ \mathcal{L}] \in \vartheta$
(Sig)	$\rho \models \mathbf{S}_{v_0^-}(v_1, \ldots, v_k)^l[dest\ \mathcal{L}] : \vartheta$	iff	$\wedge^k_{i=0} \rho \models v_i : \vartheta_i \wedge$ $\forall w_0, w_1, \ldots, w_k : \wedge^k_{i=0} w_i \in \vartheta_i \Rightarrow$ $\mathbf{S}_{w_0^-}(w_1, \ldots, w_k)^l[dest\ \mathcal{L}] \in \vartheta$
(Fun)	$\rho \models f(v_1, \ldots, v_k) : \vartheta$	iff	$\wedge^k_{i=1} \rho \models v_i : \vartheta_i \wedge$ $\forall w_1, \ldots, w_k : \wedge^k_{i=1} w_i \in \vartheta_i \Rightarrow$ $f(w_1, \ldots, w_k) \in \vartheta$
(Out)	$\rho, \kappa \models \langle e_1, \ldots, e_k \rangle.P : \psi$	iff	$\wedge^k_{i=1} \rho \models e_i : \vartheta_i \wedge$ $\forall w_1, \ldots, w_k : \wedge^k_{i=1} w_i \in \vartheta_i \Rightarrow ($ $\langle w_1, \ldots, w_k \rangle \in \kappa \wedge$ $\rho, \kappa \models P : \psi)$
(Inp)	$\rho, \kappa \models (p_1, \ldots, p_k).P : \psi$	iff	$\rho \models_1 p_1, \ldots, p_k : \kappa \triangleright \hat{W} : \psi \wedge$ $\hat{W} \neq \emptyset \Rightarrow \rho, \kappa \models P : \psi$
(New)	$\rho, \kappa \models (\nu\ n)P : \psi$	iff	$\rho, \kappa \models P : \psi$
(ANew)	$\rho, \kappa \models (\nu_\pm\ n)P : \psi$	iff	$\rho, \kappa \models P : \psi$
(Rep)	$\rho, \kappa \models !P : \psi$	iff	$\rho, \kappa \models P : \psi$
(Par)	$\rho, \kappa \models P_1 \mid P_2 : \psi$	iff	$\rho, \kappa \models P_1 : \psi \ \wedge\ \rho, \kappa \models P_2 : \psi$
(Chs)	$\rho, \kappa \models P_1\ +\ P_2 : \psi$	iff	$\rho, \kappa \models P_1 : \psi \ \wedge\ \rho, \kappa \models P_2 : \psi$
(Nil)	$\rho, \kappa \models 0 : \psi$	iff	*true*

Table 14. The Dolev-Yao Attacker.

(1)	$\wedge_{k \in \mathcal{A}_\kappa}\ \forall \langle v_1, \ldots, v_k \rangle \in \kappa :\ \wedge^k_{i=1} v_i \in \rho(z_\bullet)$
(2)	$\wedge_{k \in \mathcal{A}_{Enc}}\ \forall \{v_1, \ldots, v_k\}^l_{v_0} l[dest\ \mathcal{L}] \in \rho(z_\bullet) :$ $v_0 \in \rho(z_\bullet) \Rightarrow (\wedge^k_{i=1} v_i \in \rho(z_\bullet) \wedge (l, l_\bullet) \in \psi)$
(3)	$\wedge_{k \in \mathcal{A}_{Enc}}\ \forall v_0, v_1, \ldots, v_k :\ \wedge^k_{i=0} v_i \in \rho(z_\bullet) \Rightarrow \{v_1, \ldots, v_k\}^{l_\bullet}_{v_0}[dest\ \{l_\bullet\}] \in \rho(z_\bullet)$
(4)	$\wedge_{k \in \mathcal{A}_\kappa}\ \forall v_1, \ldots, v_k :\ \wedge^k_{i=1} v_i \in \rho(z_\bullet) \Rightarrow \langle v_1, \ldots, v_k \rangle \in \kappa$
(5)	$\{n_\bullet\} \cup \mathcal{N}_f \subseteq \rho(z_\bullet)$

4.3. Properties of the Analysis

We establish the formal correctness of our analysis by showing a subject reduction theorem and an adequacy theorem. The former ensures that the analysis information is preserved during evaluation and the latter gives an example of the semantic consequences that can be read off from the analysis.

In Subsection 2.3, we gave the definition of the pattern matching function, $\mathcal{M}$, which makes use of the substitution, $p\sigma$. The definition gives rise to the following auxiliary lemmas.

Lemma 1. if $\rho \models e : \vartheta$ and $v \in \rho(x)$ then $\rho \models e[x \mapsto v] : \vartheta$

Proof. By induction on the structure of e. □

Lemma 2. if $\rho, \kappa \models P$ and $v \in \rho(x)$ then $\rho, \kappa \models P[x \mapsto v]$

Proof. By applying the induction hypothesis on any subservices and Lemma 1 on any subexpressions. □

Lemma 3 (Substitution result). $\forall \vec{v}, \vec{p}, \sigma : \mathcal{M}(\vec{v}, \vec{p}) = \sigma$ iff $(\vec{v}) = (\vec{p})\sigma$.

Proof. By induction on the structure of $\mathcal{M}(\vec{v}, \vec{p})$. □

Lemma 4 (Pattern matching result). if $\mathcal{M}(\vec{v}, \vec{p}) = \sigma$ and $\vec{v} \in \hat{V}$ then $\rho \models_1 \vec{p} : \hat{V} \rhd \hat{W}$ and $\vec{p}\sigma \in \hat{W}$.

Proof. By induction on the structure of $\rho \models_1 \vec{p} : \hat{V} \rhd \hat{W}$. □

Lemma 5. Let P, P_1, P_2 be services. The following statements hold:

1. if $\rho, \kappa \models P_1$ and $P_1 \xrightarrow{\langle r, v_1, \ldots, v_k \rangle} P_2$ then $\rho, \kappa \models P_2$ and $\langle r, v_1, \ldots, v_k \rangle \in \kappa$
2. if $\rho, \kappa \models P_1$ and $P_1 \xrightarrow{(r, v_1, \ldots, v_k)} P_2$ and $\langle r, v_1, \ldots, v_k \rangle \in \kappa$ then $\rho, \kappa \models P_2$
3. if $\rho, \kappa \models P_1$ and $P_1 \xrightarrow{\langle r, \mathbf{E}_w \langle \vec{v} \rangle \rangle} P_2$ then $\rho, \kappa \models P_2$ and $\langle r, \{\vec{v}\}_w \rangle \in \kappa$
4. if $\rho, \kappa \models P_1$ and $P_1 \xrightarrow{(r, \mathbf{E}_w (\vec{v}))} P_2$ and $\langle r, \{\vec{v}\}_w \rangle \in \kappa$ then $\rho, \kappa \models P_2$

Proof. All four parts of Lemma 5 are proved by induction on the inference tree used to establish the semantics reduction.

We start with the base case of part 1, that is, rule (Out). Here we consider the case $\langle r, v_1, \ldots, v_k \rangle.P \xrightarrow{r, \langle \vec{v} \rangle} P$ such that $\rho, \kappa \models \langle r, \vec{v} \rangle.P$, which gives us $\langle r, \vec{v} \rangle \in \kappa$ and $\rho, \kappa \models P$ according to the analysis rule (Out).

When we prove part 2 of Lemma 5, it suffices to concentrate on the base case, that is, rule (Inp). So assume

$$\rho, \kappa \models (r, \vec{p}).P \tag{1}$$

$$(r, \vec{p}).P \xrightarrow{(r, \vec{v})} P\sigma \tag{2}$$

$$\langle r, \vec{v} \rangle \in \kappa \tag{3}$$

(2) gives $\mathcal{M}((r, \vec{v}), (r, \vec{p})) = \sigma$. Applying Lemma 4 to $\mathcal{M}((r, \vec{v}), (r, \vec{p})) = \sigma$ and (3), we have $\rho \models_1 (r, \vec{p}) : \kappa \rhd \hat{W}$ and $(r, \vec{p})\sigma \in \hat{W}$, which means that $\hat{W} \neq \emptyset$. This together with the analysis rule (Inp) gives us the expected result.

Now we shall prove part 3. We assume the following hold

$$r : w \blacktriangleright \langle r, \vec{v} \rangle.P \xrightarrow{\langle r, \mathbf{E}_w (\vec{v}) \rangle} r : w \blacktriangleright P \tag{4}$$

$$\rho, \kappa \models r : w \blacktriangleright \langle r, \vec{v} \rangle.P \tag{5}$$

Assumption (4) gives $\langle r, \vec{v} \rangle.P \xrightarrow{\langle r, \vec{v} \rangle} P$. According to Table 11, we have

$$r : w \blacktriangleright \langle r, \vec{v} \rangle.P \rightsquigarrow \langle r, \{w\}_{\vec{v}} \rangle.r : w \blacktriangleright P \tag{6}$$

(5) together with (6) give $\rho, \kappa \models \langle r, \{\vec{v}\}_w\rangle.r : w \blacktriangleright P$. Applying the analysis rule (Out) then gives the expected result.

Proving part 4 is similar. $\square$

Theorem 1 (Subject reduction). If $P_1 \xrightarrow{\tau} P_2$ and $\rho, \kappa \models P_1$ then $\rho, \kappa \models P_2$.

Proof. The proof is by induction on the inference of $P_1 \xrightarrow{\tau} P_2$ and makes use of Lemma 5.

Consider the case (t-sync) and assume the following conditions hold:

$$P_1 \xrightarrow{\langle r, \mathbf{E}_w\langle \vec{v}\rangle\rangle} P_1' \tag{7}$$

$$P_2 \xrightarrow{(r, \mathbf{E}_w(\vec{v}))} P_2' \tag{8}$$

$$\rho, \kappa \models P_1 | P_2 \tag{9}$$

The assumptions (7) and (8) give $P_1|P_2 \xrightarrow{\tau} P_1'|P_2'$. Applying the analysis rule (Par) to (9), we get $\rho, \kappa \models P_1$ and $\rho, \kappa \models P_2$. Lemma 5 then gives $\rho, \kappa \models P_1'$ and $\rho, \kappa \models P_2'$. We get the desired result $\rho, \kappa \models P_1'|P_2'$.

The remaining cases are similar or straightforward. $\square$

We shall now present an adequacy result; it focuses on the the κ component of the analysis and shows that it correctly captures the information communicated.

Theorem 2 (Adequacy). If $P_1 \xrightarrow{\tau}^* P_{n+1}$ and $\rho, \kappa \models P_1$ and at some point the value $\langle w_1, \ldots, w_k\rangle$ is output then $\langle w_1, \ldots, w_k\rangle \in \kappa$.

Proof. We may assume without loss of generality that $\langle w_1, \ldots, w_k\rangle$ is output as part of the transition $P_n \xrightarrow{\tau} P_{n+1}$. It is an easy induction on n to use Theorem 1 to show that $\rho, \kappa \models P_n$. The proof proceed by induction on the inference of $P_n \xrightarrow{\tau} P_{n+1}$ and the interesting case is (t-sync) as in the proof of Theorem 1. Thanks to Lemma 5 we have that $\langle w_1, \ldots, w_k\rangle \in \kappa$. $\square$

Similar adequacy results can be formulated for other applications of the analysis. This is perhaps a place where the development of analyses by Flow Logic differs a bit from the development of analysis by type systems. In the latter case there often is only one purpose of the type system and hence there only is one adequacy result that it is natural to formulate. In the former case the analysis tracks so many components that quite a few adequacy results can be formulated (as illustrated in [NNP11]); usually we focus on the communications taking place as established by Theorem 2.

4.4. The Analysis of the Credit Request Case Study

Returning to the Credit Request case study let us consider the general scenario that a number of *Clients* may simultaneously request services from *VS* and Ser_E (or Ser_C). We shall use i to refer to the instance of the protocol where the i'th *Client* is communicating with *VS* and Ser_E. The index i is added to all variables, crypto-points and constants thereby allowing the analysis to distinguish between the various instances.

The overall scenario of the case study takes the form

$$\begin{array}{l} ((\nu\ K^-_{VS})(\nu\ K^-_{Ser_E})(\nu\ K^-_{Ser_C}) \\ \qquad |_{i=1}^{n}\ Client_i \mid VS \mid SM \mid Ser_E \mid Ser_C) \\ \mid\ Attacker \end{array}$$

Table 15. Concrete level specification of the Credit Request Case Study.

$$
\begin{array}{l}
\text{let } X \subseteq \{1,2\} \text{ in} \\
(|_{i \in X} (\nu_{\pm}\ Kca)(\nu_{\pm}\ K_{Ser_E})(\nu_{\pm}\ Kvs)(\\
\quad /*\ \textit{Client}\ */ \\
((\nu\ Bta_i)(\nu\ r1_i)\langle r_{env}, req, r_1\rangle. \\
\quad (\nu\ Nx_i)\langle r_1, C_i, VS, Nx_i\rangle. \\
\quad\quad (r_1, VS, C_i, ?ny_i, \underline{\mathbf{S}}_{Kca^+}(VS, ?kvs) : [\mathsf{at}\ a1_i\ \mathsf{orig}\ \{b1_i\}]). \\
\quad\quad\quad (\nu\ N_i)\langle r_1, C_i, VS, \mathbf{P}_{Kvs^+}(N_i) : [\mathsf{at}\ a2_i\ \mathsf{dest}\ \{b2_i\}]\rangle. \\
\quad\quad\quad\quad r_1 : \mathbf{H}(Nx_i, ny_i, N_i) \blacktriangleright \langle r_1, Bta_i\rangle[\mathsf{at}\ a3_i\ \mathsf{dest}\ \{b3_i\}]. \\
\quad\quad\quad\quad\quad (r_1, Bta_i, ?xr_i)[\mathsf{at}\ a4_i\ \mathsf{orig}\ \{b6_i, b9_i\}].\langle r_{env}, xr_i\rangle.0) \\
|\ \ /*\ \textit{VS}\ */ \\
(r_{env}, req, ?r'_1)((r'_1, C_i, VS, ?nx_i).(\nu\ Ny_i) \\
\quad \langle r'_1, VS, C_i, Ny_i, \mathbf{S}_{Kca^-}(VS, Kvs^+) : [\mathsf{at}\ b1_i\ \mathsf{dest}\ \{a1_i\}]\rangle. \\
\quad\quad (r'_1, C_i, VS, \underline{\mathbf{P}}_{Kvs^-}(?n_i) : [\mathsf{at}\ b2_i\ \mathsf{orig}\ \{a2_i\}]). \\
\quad\quad\quad r'_1 : \mathbf{H}(nx_i, Ny_i, n_i) \blacktriangleright (r'_1, ?bta_i)[\mathsf{at}\ b3_i\ \mathsf{orig}\ \{a3_i\}]. \\
\quad\quad\quad\quad ((\nu\ r_2)\langle r_{env}, val_E, r_2\rangle.(\nu\ SN_i) \\
\quad\quad\quad\quad \langle r_2\, VS, SM, VS, Ser_E, \\
\quad\quad\quad\quad\quad\quad\quad \mathbf{S}_{Kvs^-}(\mathbf{P}_{K^+_{Ser_E}}(SN_i, bta, \mathbf{S}_{Kca^-}(VS, Kvs^+)) : [\mathsf{at}\ b4_i\ \mathsf{dest}\ \{c1_i\}])\rangle. \\
\quad\quad\quad\quad\quad (r_2, SM, VS, Ser_E, VS, \underline{\mathbf{S}}_{K^+_{Ser_E}}(\mathbf{P}_{Kvs^+}(SN_i, ?vr_i) : [\mathsf{at}\ b5_i\ \mathsf{orig}\ \{c2_i\}])). \\
\quad\quad\quad\quad\quad\quad \langle r'_1, bta_i, vr_i\rangle[\mathsf{at}\ b6_i\ \mathsf{dest}\ \{a4_i\}].0+ \\
\quad\quad\quad\quad (\nu\ r_2)\langle r_{env}, val_C, r_2\rangle.(\nu\ SN_i) \\
\quad\quad\quad\quad \langle r_2\, VS, SM, VS, Ser_C, \\
\quad\quad\quad\quad\quad\quad\quad \mathbf{S}_{Kvs^-}(\mathbf{P}_{K^+_{Ser_C}}(SN_i, bta, \mathbf{S}_{Kca^-}(VS, Kvs^+)) : [\mathsf{at}\ b7_i\ \mathsf{dest}\ \{d1_i\}])\rangle. \\
\quad\quad\quad\quad\quad (r_2, SM, VS, Ser_C, VS, \underline{\mathbf{S}}_{K^+_{Ser_C}}(\mathbf{P}_{Kvs^+}(SN_i, ?vr_i) : [\mathsf{at}\ b8_i\ \mathsf{orig}\ \{d2_i\}])). \\
\quad\quad\quad\quad\quad\quad \langle r'_1, bta_i, vr_i\rangle : [\mathsf{at}\ b9_i\ \mathsf{dest}\ \{a4_i\}].0) \\
|\ \ /*\ \textit{SM}\ */ \\
(?r, VS, SM, VS, ?ser, ?x1).\langle r, SM, ser, VS, ser, x1\rangle. \\
\quad\quad (?r', ?ser, SM, ser, VS, ?x2).\langle r', SM, VS, ser, VS, x2\rangle.0 \\
|\ \ /*\ Ser_E\ */ \\
(r_{env}, val_E, ?r'_2). \\
(r'_2, SM, Ser_E, VS, Ser_E, \\
\underline{\mathbf{S}}_{Kvs^+}(\underline{\mathbf{P}}_{K^-_{Ser_E}}(?sn_i, ?btas_i, \underline{\mathbf{S}}_{Kca^+}(VS, ?kvs)) : [\mathsf{at}\ c1_i\ \mathsf{orig}\ \{b4_i\}])). \\
\langle r'_2, Ser_E, SM, Ser_E, VS, \mathbf{S}_{K^-_{Ser_E}}(\mathbf{P}_{kvs^+}(sn_i, isValid(btas_i)) : [\mathsf{at}\ c2_i\ \mathsf{dest}\ \{b5_i\}])\rangle.0 \\
|\ \ /*\ Ser_C\ */ \\
(r_{env}, val_C, ?r'_2). \\
(r'_2, SM, Ser_C, VS, Ser_C, \\
\underline{\mathbf{S}}_{Kvs^+}(\underline{\mathbf{P}}_{K^-_{Ser_C}}(?sn_i, ?btas_i, \underline{\mathbf{S}}_{Kca^+}(VS, ?kvs)) : [\mathsf{at}\ d1_i\ \mathsf{orig}\ \{b7_i\}])). \\
\langle r'_2, Ser_C, SM, Ser_C, VS, \mathbf{S}_{K^-_{Ser_C}}(\mathbf{P}_{kvs^+}(sn_i, isValid(btas_i)) : [\mathsf{at}\ d2_i\ \mathsf{dest}\ \{b8_i\}])\rangle.0))
\end{array}
$$

reflecting the fact that *VS*, *SM*, Ser_E and Ser_C are ready to interact with legitimate *Clients* as well as the attacker, and the attacker has no knowledge of the principals' private keys. One may also include a dishonest *Client*, who shares long-term keys with the *Certificate Authority* (*CA*). However such an inside attacker is so powerful in this example that he is able to interfere with almost all the communications and we therefore concentrated on the outside attacker. The analysis itself is carried out for $n = 2$, which amounts to partitioning the *infinite* number of *Clients* into two groups, with each group communicating with the rest of the principals. This allows the analysis to determine if any two instances of communications can interfere with each other. The concrete level specification (with LySa annotations) of the Credit Request case study is shown in Table 15.

In case the unilateral *TLS* authentication protocol is adopted (for protecting the communications between *Client* and *VS*) the analysis gives an empty ψ component (i.e. $\psi = \emptyset$) which means that no authentication annotations are violated. For example, one can draw the conclusion that *once the decision has been made whether the request has to be validated by the service for enterprises or corporates, it cannot be tricked into being processed by the wrong one*, because $(b4_i, d1_i) \notin \psi$. Inspecting the analysis result more closely, the following entries may be of interest:

$$\begin{array}{l}\rho(x_{r1}) = \{\mathit{isValid}(Bta_1)\} \\ \rho(x_{r2}) = \{\mathit{isValid}(Bta_2)\}\end{array}$$

This confirms that the evaluation results $\mathit{isValid}(Bta_i)$ are correctly returned back from Ser_E to *Client*, via *VS* and *SM*. Furthermore, the analysis results also show that no sensitive date is leaked to the attacker, i.e. the attacker's knowledge $\rho(z_\bullet)$ does not contain any important information (recall that $z_\bullet$ is the variable used by the attacker). In summary, both authentication and confidentiality hold in this case study.

4.5. The Analysis of the SAML Single Sign-On Protocol

We consider the scenario where there are a number of users and destinations, but only one server. We shall use i and j to refer to an instance of the protocol involving the i'th user and j'th destination. The indices are added to the names of the principals, variables, constants and crypto-points, thus allowing different instances to be distinguished. We shall use the index 0 to refer to the dishonest user, which shares a long term key with the server. The communications between each two parties, user and server, user and destination, as well as server and destination, are all protected by *TLS* unilateral authentication protocol. The overall scenario takes the form:

$$(|_{i=0}^{n}|_{j=1}^{m}(\nu\ Ks^-)(\nu\ K_{Ui})\ U_i \mid S \mid D_j) \mid \mathit{Attacker}$$

The concrete level specification (with LySa annotations) of the SAML Single Sign-On Protocol is shown in Table 16. (The two underlines are irrelevant at this point and shall be referred to later on.)

The analysis is carried out for $n = 2$ and $m = 2$, which models that two groups of users and two groups of destinations are communicating with each other via one server S. The analysis result has a non-empty error component $(c4_{ij}, l_\bullet)$, which means that the message M sent by the destination to the user in step 6 can be learnt by the attacker (recall that $l_\bullet$ is the crypto-point used by the attacker). The ρ component contains the entry $K_{Ui} \in \rho(z_\bullet)$, which is relevant to this situation.

This shows that the attacker can also learn the master key K_{Ui}. This information leads us to find the following attack:

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

In the above attack, the attacker first pretends to be the user U and acquires an Artifact from the server S in step 1 and 2. In step 4, it sends the Artifact back to S pretending to be a destination D. Finally in step 5, the key K_U is compromised. Now the attacker is able to decrypt the message M that is encrypted using K_U.

This attack is not surprising; the communications are only secured by the *TLS* unilateral authentication protocol. The attack can be avoided by adopting the *TLS* bilateral protocol to protect the communication between the user and the server. In this case, the abstract level specification remains the same, and the plug-in level specification becomes:

Table 16. Concrete level specification of the SAML Single Sign-On Protocol.

$$\begin{array}{l}
\text{let } X \subseteq \{0,1,2\}, Y \subseteq \{1,2\} \text{ in} \\
(|_{i\in X, j\in Y} (\nu_{\pm}\ Kca)(\nu_{\pm}\ Ks)(\nu_{\pm}\ Ku_i)(\nu\ Kd_j)(\\
\quad /*U*/ \\
(\nu\ r_1)\langle r_{env}, inv, r_1\rangle. \\
\ (\nu\ Nu1_{ij})\langle r_1, U_i, S, Nu1_{ij}\rangle.(r_1, S, U_i, ?ns1_{ij}, \underline{\mathbf{S}}_{K^+_{CA}}(S, ?xks1_{ij})). \\
\quad (\nu\ N1_{ij})\langle r_1, U_i, S, \mathbf{P}_{xks1_{ij}}(N1_{ij})\rangle.r_1 : \mathbf{H}(Nu1_{ij}, ns1_{ij}, N1_{ij}) \blacktriangleright (\\
\quad\ \langle r_1, U_i, \overline{S, D_j\rangle[\text{at } a1_{ij} \text{ dest } \{b1_{ij}\}}].(r_1, S, U_i, D_j, ?a_{ij})[\text{at } a2_{ij} \text{ orig } \{b2_{ij}\}]. \\
\quad\quad (\nu\ r_2)\langle r_{env}, fwd, r_2\rangle. \\
\quad\quad\ (\nu\ Nu2_{ij})\langle r_2, U_i, D_j, Nu2_{ij}\rangle.(r_2, D_j, U_i, ?ns2_{ij}, \underline{\mathbf{S}}_{K^+_{CA}}(D_j, ?xks2_{ij})). \\
\quad\quad\ (\nu\ N2_{ij})\langle r_2, U_i, D_j, \mathbf{P}_{xks2_{ij}}(N2_{ij})\rangle. \\
\quad\quad\quad r_2 : \mathbf{H}(Nu2_{ij}, ns2_{ij}, N2_{ij}) \blacktriangleright (\\
\quad\quad\quad \langle r_2, U_i, D_j, S, a_{ij}\rangle[\text{at } a3_{ij} \text{ dest } \{c1ij\}]. \\
\quad\quad\quad\ (r_{env}, send, ?r'_4).r'_4 : Ku_i \blacktriangleright (r'_4, ?m_{ij})[\text{at } a4_{ij} \text{ orig } \{c4_{ij}\}].0)) \\
\\
|\quad /*S*/ \\
(r_{env}, inv, ?r'_1).(r'_1, U_i, S, ?nu1_{ij}).(\nu\ Ns1_{ij})\langle r'_1, S, U_i, Ns1_{ij}, \mathbf{S}_{K^-_{CA}}(S, K^+_S)\rangle \\
\ (r'_1, U_i, S, \underline{\mathbf{P}}_{K^-_S}(?n1_{ij})). \\
\quad \overline{r'_1 : \mathbf{H}(nu1_{ij}, Ns1_{ij}, n1}_{ij}) \blacktriangleright (\\
\quad\ (r'_1, U_i, S, ?d)[\text{at } b1_{ij} \text{ orig } \{a1_{ij}\}].(\nu\ A_{ij})\langle r'_1, S, U_i, d, A_{ij}\rangle[\text{at } b2_{ij} \text{ dest } \{a3_{ij}\}]. \\
\quad\quad (r_{env}, val, ?r'_3).(r'_3, D_j, S, ?nd3_{ij}).(\nu\ Ns3_{ij})\langle r'_3, S, D_j, Ns3_{ij}, \mathbf{S}_{K^-_{CA}}(S, K^+_S)\rangle \\
\quad\quad\ (r'_3, D_j, S, \underline{\mathbf{P}}_{K^-_S}(?n3_{ij})). \\
\quad\quad\quad r'_3 : \mathbf{H}(nd3_{ij}, Ns3_{ij}, n3_{ij}) \blacktriangleright (\\
\quad\quad\quad\ (r'_3, d, S, A)[\text{at } b3_{ij} \text{ orig } \{c2_{ij}\}].\langle r'_3, S, d, Ku_i, U_i\rangle[\text{at } b4_{ij} \text{ dest } \{c3_{ij}\}].0)) \\
|\quad /*D*/ \\
(r_{env}, fwd, ?r'_2).(r'_2, U, D, ?nu2_{ij}). \\
\ (\nu\ Nd2_{ij})\langle r'_2, D_j, U_i, Nd2_{ij}, \mathbf{S}_{K^-_{CA}}(S, K^+_S)\rangle \\
\quad (r'_2, U_i, D_j, \underline{\mathbf{P}}_{K^-_S}(?n2_{ij})). \\
\quad\ r'_2 : \mathbf{H}(nu2_{ij}, Nd2_{ij}, n2_{ij}) \blacktriangleright (\\
\quad\quad (r'_2, U_i, D_j, S, ?a'_{ij})[\text{at } c1_{ij} \text{ orig } \{a3_{ij}\}]. \\
\quad\quad\ (\nu\ r_3)(r_{env}, val, r_3).(\nu\ Nd_{ij})\langle r_3, D_i, S, Nd_{ij}\rangle. \\
\quad\quad\quad (r_3, S, D_i, ?ns3_{ij}, \underline{\mathbf{S}}_{K^+_{CA}}(S, ?xks3_{ij})) \\
\quad\quad\quad\ (\nu\ N3_{ij})\langle r_3, D_i, S, \mathbf{P}_{xks3_{ij}}(N3_{ij})\rangle. \\
\quad\quad\quad\quad r_3 : \mathbf{H}(Nd_{ij}, ns3_{ij}, N3_{ij}) \blacktriangleright (\\
\quad\quad\quad\quad\ \langle r_3, D_j, S, a'_{ij}\rangle[\text{at } c2_{ij} \text{ dest } \{b3_{ij}\}].(r_3, S, D_j, ?ku_{ij}, U_i)[\text{at } c3_{ij} \text{ orig } \{b4_{ij}\}]. \\
\quad\quad\quad\quad\quad (\nu\ M_{ij})(\nu\ r_4)\langle r_{env}, send, r_4\rangle.ku_{ij} \blacktriangleright (\\
\quad\quad\quad\quad\quad\ \langle r_4, M_{ij}\rangle[\text{at } c4_{ij} \text{ dest } \{a4_{ij}\}].0)))))
\end{array}$$

$$\begin{array}{rcl}
U & \triangleq & \overline{inv\nu}[\mathbf{TLS_2}, U, S, CA].\langle U, S, D\rangle.(S, U, D, ?a). \\
 & & \quad \overline{fwd\nu}[TLS, U, D, CA].\langle U, D, S, a\rangle. \\
 & & \quad\quad send\nu[enc, K_U].(?m).0 \\
\\
S & \triangleq & inv\nu[\mathbf{TLS_2}, U, S, CA].(U, S, ?d)(\nu\ A)\langle S, U, d, A\rangle. \\
 & & \quad val\nu[TLS, D, S, CA].(d, S, A).\langle S, d, K_U, U\rangle.0 \\
\\
D & \triangleq & fwd\nu[TLS, U, D, CA].(U, D, S, ?a'). \\
 & & \quad \overline{val\nu}[TLS, D, S, CA].\langle D, S, a'\rangle.(S, D, ?k_U, U). \\
 & & \quad\quad (\nu\ M)\overline{send\nu}[enc, k_U]\langle M\rangle.0
\end{array}$$

The only changes that have to made are replacing TLS by TLS_2 in the invocation and response of the service Inv, as indicated by our use of **bold** font.

The concrete specification is the same as in Table 16 except that the output and input, marked with underlines, become

$$\langle r_1, U_i, S, \mathbf{S}_{K^-_{CA}}(C, K^+_C), \mathbf{P}_{xks1_{ij}}(N1_{ij}), \mathbf{S}_{K^-_{Ui}}(Nu1_{ij}, ns1_{ij}, N1_{ij}))\rangle$$

and

$$(r'_1, U_i, S, \underline{\mathbf{S}}_{K^+_{CA}}(C, ?kc)\underline{\mathbf{P}}_{K^-_S}(?n1_{ij}), \underline{\mathbf{S}}_{kc}(nu1_{ij}, Ns1_{ij}, n1_{ij}))$$

Applying the analysis to the new version of the case study now gives an empty error component, i.e. $\psi = \emptyset$. This result verifies that the attack cannot happen when the communications between the user and the server are secured by the *TLS* bilateral authentication protocol.

Our work here is based on version 1.0 of the SAML specification. The work most closely related to ours is [HSN05], which also proves the existence of a similar attack, and [ACC$^+$08] that proves an attack on Google's implementation of SAML. However, later work [ACC$^+$08] seems to show that version 2.0 of the SAML specification does no longer admit such an attack, as the new specification provides a more detailed treatment of the security aspects of the protocol.

5. Conclusion

This paper presented the CaPiTo framework, which is able to model service-oriented systems at different levels of abstractions, with or without taking the underlying protocol stack into consideration. To the best of our knowledge this is a novel contribution.

We formally developed the *abstract*, the *plug-in* and the *concrete* levels of CaPiTo together with the semantics of the concrete level and showed how to transform the plug-in level to the concrete level. We also developed a static analysis to formally track the run-time behaviour and check the authentication properties of systems. As usual our analysis has been implemented using our LySa technology [BBD$^+$05] and in general the results are computed in low polynomial time in the size of the system in question. In practice the analysis is quite fast; for each experiment that we have conducted, the analysis is able to give result within two seconds. Throughout the paper we illustrated our approach on the Credit Request case study developed and scrutinised as part of the EU project SENSORIA [Sen]. Although LySa is used as backend in this work, it is also possible to replace it by other protocol verification tools, e.g. ProVerif [Pro], Scyther [Scy], or OFMC [MV09].

We chose process algebra as the way to model service-oriented applications, as it has been widely used for describing concurrent communicating systems. There are many other ways of modelling systems, visual representations for example, that the designers of computer systems are more familiar with. However, not all the visual notations, e.g. state diagrams, flow charts, Petri nets, etc, have a universally accepted underlying semantics and are associated with formal analysis techniques [Rei85]. Process algebras, on the other hand, are equipped with operational semantics and therefore subject to formal analysis. They also provide succinct yet precise descriptions of communication systems. Furthermore, for industrial purposes, it is also possible to embed process algebra in high level programming languages, e.g. C++ or Java. There are also tools available for facilitating non-experts in formal methods. For example, the twin tools Elyjah [O'S] and Hajyle translate between LySa processes and Java code; this helps developers to increase their understanding of security protocols and also helps them to develop more secure programs.

In our view the main contribution of the CaPiTo approach, compared to that of other service-oriented calculi (e.g. CaSPiS [BBNL08]), is that the CaPiTo approach, on the one hand, allows to perform an abstract modelling of service-oriented applications and, on the other hand, facilitates dealing with existing standardised protocols. It is due to this ability that we believe CaPiTo overcomes a shortcoming identified in the EU project SENSORIA — that there is a gap between the level of models and analyses performed by the academic partners and the realisations and implementations performed by the industrial partners.

From a more theoretical perspective we could equip the abstract level of CaPiTo with a semantics in the same style as the one given to CaSPiS [BBNL08] and we could then study equivalences between specifications at the various levels of CaPiTo. Similarly we could develop static analyses at several levels of CaPiTo and compare their relative precision. However, in our view this is not what the industrial partners are likely to find useful; rather we believe that an analysis performed as close as possible to the concrete specification level is more valuable in practice. Indeed, it reduces the risks of attacks at levels below the level of formalisation.

Acknowledgment. We should like to thank Chiara Bodei for working with us in the Sensoria project, Jose Nuno Carvalho Quaresma and the referees for providing useful comments on improving the presentation of the paper, and Ender Yuksel for assistance with the typesetting.

References

[ACC$^+$08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, Hilton Alexandria Mark Center, Virginia, USA, 2008. ACM Press.

[AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[BBD$^+$05] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static validation of security protocols. *J. Comput. Secur.*, 13:347–390, May 2005.

[BBNL08] Michele Boreale, Roberto Bruni, Rocco Nicola, and Michele Loreti. Sessions and pipelines for structured service programming. In *Proceedings of the 10th IFIP WG 6.1 international conference on Formal Methods for Open Object-Based Distributed Systems*, FMOODS '08, pages 19–38, Berlin, Heidelberg, 2008. Springer-Verlag.

[DA99] T. Dierks and C. Allen. The tls protocol version 1.0, 1999.

[Gao08] Han Gao. *Analysis of security protocols by annotations.* PhD thesis, Technical University of Denmark, 2008.

[GNN11] Han Gao, Flemming Nielson, and Hanne Riis Nielson. Analysing Protocol Stacks for Services. In *Rigorous Software Engineering for Service-Oriented Systems*, page to appear. Springer LNCS, 2011.

[HM04] J. Hughes and E. Maler. Security assertion markup language (SAML) v1.1 technical overview. Technical report, 2004.

[HSN05] Steffen M. Hansen, Jakob Skriver, and Hanne Riis Nielson. Using static analysis to validate the saml single sign-on protocol. In *Proceedings of the 2005 workshop on Issues in the theory of security*, WITS '05, pages 27–40, New York, NY, USA, 2005. ACM.

[LG00] Guy Leduc and Franois Germeau. Verification of security protocols using lotos - method and application. *Computer Communications*, 23:2000, 2000.

[Lib] Liberty alliance project.

[LPT07] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A calculus for orchestration of web services. In *Proceedings of the 16th European conference on Programming*, ESOP'07, pages 33–47, Berlin, Heidelberg, 2007. Springer-Verlag.

[Mil99] Robin Milner. *Communicating and mobile systems: the &pgr;-calculus.* Cambridge University Press, New York, NY, USA, 1999.

[MV09] Sebastian Mödersheim and Luca Viganò. *The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols*, pages 166–194. Springer-Verlag, Berlin, Heidelberg, 2009.

[NAPN] Christoffer Rosenkilde Nielsen, Michel Alessandrini, Michael Pollmeier, and Hanne Riis Nielson. Formalising the S&N Credit Request. Technical report, Technical University of Denmark, Informatics and Mathematical Modelling, Technical University.

[NNP11] Hanne Riis Nielson, Flemming Nielson, and Henrik Pilegaard. Flow logic for process calculi. *ACM Computing Surveys*, page to appear, 2011.

[NNS02] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. A succinct solver for alfp. *Nordic J. of Computing*, 9:335–372, December 2002.

[OAS] Organization for the advancement of structured information standards.

[O'S] Nicholas O'Shea. The elyjah project.

[OSI] Itu-t x.200 (07/94) the basic reference model (osi).

[Pro] Proverif: Cryptographic protocol verifier in the formal model.

[Rei85] Wolfgang Reisig. *Petri nets: an introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[Scy] Scyther tool.

[Sen] Sensoria project. `http://sensoria.fast.de/`.

[Shi] Shibboleth project. `http://shibboleth.internet2.edu/index.html`.

[SOA] Simple object access protocol (soap).

[Sta99] William Stallings. *Cryptography and network security (2nd ed.): principles and practice.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[WS-] Oasis web services security (wss) tc.

Table 17. A direct definition of the pattern matching function.

$\mathcal{M}(v, v)$	$=$	ϵ
$\mathcal{M}(v, ?x)$	$=$	$[(x, v)]$
$\mathcal{M}(\epsilon, \epsilon)$	$=$	ϵ
$\mathcal{M}((v_1, \ldots, v_k), (p_1, \ldots, p_k))$	$=$	let $\sigma = \mathcal{M}(v_1, p_1)$ in $\sigma \uplus \mathcal{M}((v_2, \ldots, v_k), \sigma(p_2, \ldots, p_k))$
$\mathcal{M}(\mathbf{P}_{v^+}(v_1, \ldots, v_k), \underline{\mathbf{P}}_{v^-}(p_1, \ldots, p_k))$	$=$	$\mathcal{M}((v_1, \ldots, v_k), (p_1, \ldots, p_k))$
$\mathcal{M}(\mathbf{S}_{v^-}(v_1, \ldots, v_k), \underline{\mathbf{S}}_{v^+}(p_1, \ldots, p_k))$	$=$	$\mathcal{M}((v_1, \ldots, v_k), (p_1, \ldots, p_k))$
otherwise	$=$	*undefined*

A. The Pattern Matching Function

The definition of Pattern Matching Function $\mathcal{M}$ that leads to an implementation is listed in Table 17. The function determines whether a receive and an invoke over the same endpoint can synchronize. The definition, as shown in Table 17, states that two identical values match each other, a binding variable matches any value, a pattern tuple matches a value tuple only if they are of the same length and inductively pairwise matches, and two encryptions matches when both the encrypted messages and the keys match each other. The notation ϵ stands for the empty list. The pattern matching function returns a substitution $\sigma \in (Var \times Val)^*$ for all the binding variables. The operation $\uplus$ is used to concatenate two substitutions.