Towards Security Analyses of an Identity Federation Protocol for Web Services in Convergent Networks

Maurice ter Beek
ISTI–CNR, Via G. Moruzzi 1, 56124 Pisa, Italy
Email: maurice.terbeek@isti.cnr.it
Corrado Moiso
Telecom Italia, Via Reiss Romoli 274, 10148 Torino, Italy
Email: corrado.moiso@telecomitalia.it
Marinella Petrocchi
IIT–CNR, Via G. Moruzzi 1, 56124 Pisa, Italy
Email: marinella.petrocchi@iit.cnr.it

***Abstract*— We describe a formal approach to the analysis of security aspects of an identity federation protocol for web services in convergent networks. This network protocol was proposed by Telecom Italia as a solution to allow end users to access services on the web through different access networks without explicitly providing any credentials, while the service providers can trust the user's identity information provided by the access networks and access some user data. As a first step towards a full-blown formal security analysis of the protocol, we specify three user scenarios in the process algebra Crypto-CCS and verify the vulnerability of one of these specifications w.r.t. a man-in-the-middle attack with the model checker PaMoChSA.**

I. Introduction

Formal methods and tools are popular means for the analysis of security aspects of computer network protocols. First the protocol under scrutiny is described in a formal language, which often results in a more precise definition of its functioning. Subsequently, the security aspects to be analyzed are specified in a logic. Finally, to decide whether or not certain security properties are fulfilled by the protocol, an automatic tool is used to analyze the protocol. The outcome either proves the protocol correct w.r.t. the security aspects considered or shows how it falls to one or more attacks [1]–[7].

Research in this active branch of computer security has led to a variety of formal techniques; some of these are based on process algebras [8]–[11]; others on proof techniques for authentication logic [12]–[14]; some exploit type systems and other static analyses [15], [16]; yet others use automata [17]. A common analysis strategy is to test the behaviour of one's protocol specification within a hostile environment, by running it in a setting in which the presence of honest participants is complemented with a malicious adversary.

In this paper we formally specify three user scenarios of an identity federation protocol for (web) services in convergent networks and we analyse some of its security aspects (like the possibility of a man-in-the-middle attack). To do so we use Crypto-CCS, a CCS-like process algebra with cryptographic primitives [2], [18], [19], in combination with the Partial Model Checking Security Analyzer PaMoChSA [3], [11], [20] developed by the Security group of IIT–CNR.

The network protocol we will formally specify and analyze was proposed in [21] to permit end users to access services through different access networks (*e.g.* mobile and fixed ones) without explicitly providing any credentials, while the application level can trust the identity and authentication information provided by the access networks. As a result, service providers (SPs) identify a user by using the authentication procedure performed by the network provider. After identity federation, a single sign-on suffices for a user to access all services belonging to the same "circle of trust" of SPs while keeping personal data private. This is an advantage over introducing (and remembering) one's credentials time and again. The protocol may thus grant users anonymous access to services and at the same time allow the application level to limit access to authorized users. Moreover, without knowing who the user is, SPs can still obtain her/his location or age or charge her/his account. The protocol is related to recent solutions specified by the Liberty Alliance project [22], in which case the task to authenticate a user is delegated to an identity provider.

The paper is structured as follows. After this introduction, we recall the network protocol of [21] in detail in Section II. In Section III we then describe our analysis approach, after which we formally specify three user scenarios of the protocol in Section IV. A first security analysis of the protocol is presented in Section V. Finally, Section VI contains our conclusions and some indications for future work.

II. Protocol Description

In this section we recall the relevant features of the identity federation protocol proposed in [21] as a solution to handle the identity and authentication information of end users that access WSs on convergent networks through multiple telecommunication channels (*e.g.* ADSL, GPRS/UMTS, SMS). Providers of telecommunication networks that offer their end users access to WSs are in a privileged position to pass this information of the users that actually access their network, to the SPs. When passing information obtained within their security domain to the application level, the network providers do need to take the following security aspects into account:

1) privacy of sensitive user information (*e.g.* network address); 2) guarantee a user's identity, without explicitly discovering it.

To this aim, the protocol makes use of a token injector mechanism to translate the identity and authentication information provided by a secure access network to the Internet application level (with a lower level of security). The token injector thus plays the identity provider role as described in the Liberty Alliance specifications [23].

The token injector intercepts all the data traffic that originates from a terminal connected to the secure network and that, over an insecure network, has an application server as destination. It then adds some token carrying the identity information of the end user to the intercepted messages. The format of such tokens must be compatible with the application protocol used in the interaction and the identity information must be such that the application can use it to identify one of its users. See Figure 1 for a sketch of the described scenario.

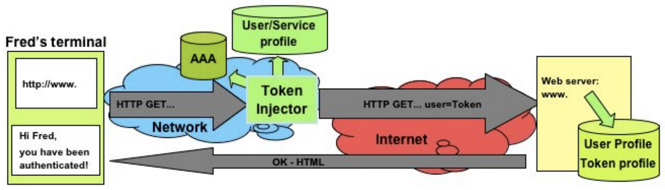


Fig. 1. Token injector mechanism.

When a user requests access to a specific WS, the token injector deployed in the network operator domain intercepts the HTTP GET (or POST) request and, depending on the information stored in the operators repositories (such as user profile, service profile, etc.), inserts a token (*i.e.* a string of characters) into the original HTTP packet and forwards it to the final destination, for instance to a SP. When the application server receives the new access request, it looks for a token. If it is present and valid (*e.g.* issued by a trusted authority, well-formed, etc.), then it immediately recognizes and welcomes the associated user without explicitly asking for credentials. The token is formatted according to the SAML standard [24] and it can carry additional information, like a description of how the end user has been authenticated. The token is protected from various security issues by means of cryptographic tools, like digital signatures and symmetric encryption to protect its integrity and confidentiality and sequence numbers against replay attacks. Likewise, communication between the token injector and the SP is by means of secure protocols, like SSL or TLS. The token injector mechanism may also be applied to add identity tokens to SOAP or SIP protocols and the like, again according to some available standard. For instance, the tokens added to SOAP messages could be formatted according to the OASIS WS-Security specification.

We now provide an example of a user that accesses a WS through multiple telecommunication channels. Consider a scenario with two network operators, a Fixed Operator FO and a Mobile Operator MO, both of which have implemented the token injector mechanism and have established a trusted relation with a SP, for instance a travel site. A user that is a customer of both operators, has an active registration on the travel site and has already federated her/his account on the travel site with her/his profile on each of the operators (*i.e.* two federations have been activated: one between the SP and the FO accounts and one between the SP and the MO accounts). During the process of federation, the token injector generates an opaque-id (or pseudonym) for the user and sends it to the SP. This opaque-id is then stored on the repositories associated to the token injector and to the SP and it is exchanged in all communication between the operators and the SP, so as to identify the user in a secure way. Consequently, the user can access the travel site by PC (ADSL) or mobile phone (GPRS) without introducing credentials. Instead, the network authentication is forwarded to the travel site, which identifies the user. Even though the SP does not know the user's mobile phone number, the opaque-id enables it to use the operators service (as SMS gateway) to nevertheless exchange or request information about the user. The architecture of the described scenario is sketched in Figure 2.

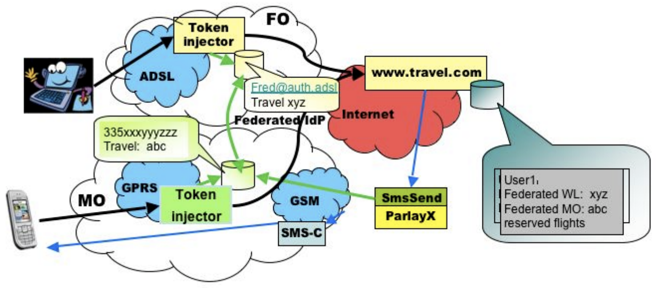


Fig. 2. Identity federation in convergent networks.

III. Analysis Approach

We adopt the analysis approach of [19]. This approach is based on the observation that a protocol under analysis can be described as an open system: a system in which some component has an unspecified behaviour (not fixed in advance). Subsequently one assumes that, regardless of the unspecified behaviour, the system works properly (*i.e.* satisfies a certain property). In case of the network protocol of the previous section, one can imagine the presence of a hostile adversary trying to interfere with the normal execution of the protocol, in order to achieve some kind of advantage w.r.t. the honest participants. Such an adversary is added to the specification of the network protocol, presented in the next section, as a component with a behaviour that is defined only implicitly by the semantics of the specification language.

We assume the adversary to act in Dolev-Yao fashion [25] by using a set of message manipulating rules that model cryptographic functions like encryption and decryption. Encryption is opaque, *i.e.* a message encrypted with the public key of one of the participants cannot be decrypted by anyone but the person who knows the corresponding private key (unless the decryption key is compromised of course). As is common in this branch of computer security, we adopt a black-box view of cryptography by assuming all cryptographic primitives involved in the network protocol to be perfect. Like the honest participants, the adversary is able to send and receive messages to other participants. However, it can also intercept and forge messages and, to a certain degree, derive new messages from the set of messages that it has come to know. This set consists of all messages the adversary knows from the beginning (its initial knowledge) united with the messages it can derive from the ones intercepted during a run of the protocol. To analyze whether a system works properly, at a certain point in the run the adversary's knowledge is checked against a security property. If the intruder has come to know information it was not supposed to know, then the analysis has thus revealed an attack w.r.t. that particular property, *i.e.* a sequence of actions performed by the adversary that invalidates the property.

As said before, we will model the network protocol in Crypto-CCS and analyze a part of it with PaMoChSA. Before describing the basics of Crypto-CCS, we fix some notation and terminology related to the security primitives we deal with.

A. Notation and Terminology

The sending of a message *msg* from sender A to receiver B over the ith communication channel c_i is denoted by

$$c_i \quad A \mapsto B \quad : \quad \mathit{msg}$$

In the sequel we will use the following security primitives:

$pk_i,\ pk_i^{-1}$	*public and private key of agent i*
$\{\_\}_{pk_i^{-1}}$	*message signed by agent i*
$\{\_\}_{pk_i}$	*message encrypted by public key of agent i*
$\{\_\}_{\mathit{KEY}}$	*message encrypted by symmetric key KEY*
n_j^i	*nonce related to j generated by i*

A *nonce* is a parameter that varies with time, *e.g.* a special marker intended to prevent the unauthorized replay or reproduction of a message. Indeed, a nonce is "generated with the purpose of being used in a single run of the protocol" [26]. Nonces are commonly implemented as pseudo-random strings.

B. Crypto-CCS

We define those parts of the syntax and semantics of Crypto-CCS [2], [18], [19] needed to model the network protocol.

Crypto-CCS is a slightly modified version of CCS [27], including some cryptographic primitives. A model defined in Crypto-CCS consists of a set of sequential agents able to communicate by exchanging messages (*e.g.* data manipulated by the agents). Inference systems model the possible operations on messages and therefore consist of a set of rules of the form:

$$r = \frac{m_1 \quad \cdots \quad m_n}{m_0}$$

where $m_1, \ldots, m_n$ form a set of premises (possibly empty) and m_0 is the conclusion. An instance of the application of a rule r to closed messages $m_1, \ldots, m_n$ (*i.e.* messages without variables) is denoted as $m_1 \quad \cdots \quad m_n \vdash_r m_0$.

The control part of the language consists of compound systems, basically sequential agents running in parallel. The terms of the language are generated by the following grammar (only constructs used in the sequel are presented):

$$\begin{array}{ll}
S := S_1 \parallel S_2 & \textit{compound systems} \\
A := \mathbf{0} \mid p.A \mid [m_1 \cdots m_n \vdash_r x]A; A_1 & \textit{sequential agents} \\
p := c!m \mid c?x & \textit{prefix constructs}
\end{array}$$

where $m_1, \ldots, m_n, m$ are closed messages or variables, x is a variable and c is an element of the finite set *Ch* of channels. Informally, the Crypto-CCS semantics used in the sequel are:

- $c!m$ denotes a message m sent over channel c;
- $c?x$ denotes a message m received over channel c which replaces the variable x;
- $\mathbf{0}$ denotes a process that does nothing;
- $p.A$ denotes a process that can perform an action according to p and then behave as A;
- $[m_1 \cdots m_n \vdash_r x]A; A_1$ denotes the inference construct: if (by applying an instance of rule r with premises $m_1, \ldots, m_n$) a message m can be inferred, then the process behaves as A (where m replaces x), otherwise it behaves as A_1. This is the message manipulating construct of the language, *e.g.* $[m \quad pk_y^{-1} \vdash_{sign} x]A; \mathbf{0}$ is a process that uses the rule $sign$ to obtain a digitally signed message from plaintext message m and private key pk_y^{-1} and then behaves as A, or otherwise does nothing;
- $S_1 \parallel S_2$ denotes the parallel composition of S_1 and S_2, *i.e.* $S_1 \parallel S_2$ performs an action if either S_1 or S_2 does. It may perform a synchronization or internal action, denoted by τ, whenever S_1 and S_2 can perform two complementary send and receive actions over the same channel.

The language is completely parametric w.r.t. the inference system used. In particular, the inference system that is used below to model the network protocol is shown in Figure 3.

$$\frac{x \quad y}{\mathit{Pair}(x,y)}\ (\mathit{pair}) \qquad \frac{x \quad pk_y^{-1}}{\{x\}_{pk_y^{-1}}}\ (\mathit{sign}) \qquad \frac{x \quad \mathit{KEY}}{\{x\}_{\mathit{KEY}}}\ (\mathit{enc})$$

$$\frac{\mathit{Pair}(x,y)}{x}\ (\mathit{1st}) \qquad \frac{\{x\}_{pk_y^{-1}} \quad pk_y}{x}\ (\mathit{ver}) \qquad \frac{\{x\}_{\mathit{KEY}} \quad \mathit{KEY}}{x}\ (\mathit{dec})$$

$$\frac{\mathit{Pair}(x,y)}{y}\ (\mathit{2nd}) \qquad \frac{x}{x}\ (\mathit{check})$$

Fig. 3. Inference system for the network protocol.

Rule (*pair*) builds the pair of two messages x and y. Rules (*1st*) and (*2nd*) return the components of a pair. Rule (*sign*) digitally signs a message x by applying the secret key pk_y^{-1} of agent y. Rule (*ver*) verifies a digital signature $\{x\}_{pk_y^{-1}}$ by applying the public key pk_y of signer y. Rule (*enc*) encrypts a message x by applying the symmetric key *KEY*. Rule (*dec*) decrypts a message $\{x\}_{\mathit{KEY}}$ by applying the symmetric key *KEY*. Finally, rule (*check*) performs checks on the correctness of authentication statements and on the freshness of nonces.

A system S of two known components S_1 and S_2 and a third unspecified component (the adversary) is thus described as $S := S_1 \parallel S_2 \parallel \_$. To verify a security property of S, one thus needs to do so w.r.t. every possible third component (adversary). In [19] a way to do this is given by extending partial model checking techniques [28].

IV. Protocol Formalization

In this section we formalize the protocol described in Section II in the approach lined out in the previous section.

A. Federated Registration

The first, basic scenario that we formalize is that of federated registration. This process is lined out in detail in Figure 4, made available by Telecom Italia. Our formalization starts with message 4: Request of registration + federation.

The entities involved in this scenario are thus a user U, a service provider *SP* and a network/identity provider *IdP*. U asks for a registration to *SP* and a federation between *SP* and *IdP* in the sense that—once federated—*IdP* and *SP* may provide U access without directly asking for any credentials, but by simply relying on the information given by *IdP*. Federated registration consists of three main phases: the authentication phase in which *IdP* authenticates U, the token generation and the assembling of a so-called *SAML response* to be sent from *IdP* to *SP*. For modelling purposes only, we identify a *SAML assertion* and a *SAML response* from now on.

The Security Assertion Markup Language SAML [24] is an XML standard to exchange information on authentication and authorization data between security domains (*e.g.* between *IdP* and *SP*) intended to implement mechanisms for single signons. A *SAML assertion* declares a certain subject authenticated by a particular means at a particular time. For our purposes, it contains a field *Subj* with the token id_U univocally identifying U, a field *Auth Stat* with an authentication statement asserting that U was authenticated as well as the mechanism under which she/he was authenticated and, finally, a field $\mathit{Attr\,Stat} = \langle \mathit{attr\,list}, n_U^{\mathit{IdP}} \rangle$ with a list of attributes of U related to her/his service accesses (*e.g.* country preferences if the service is a travel agency) and a nonce to avoid replay attacks:

$$\{\mathit{Subj}, \mathit{Auth\,Stat}, \mathit{Attr\,Stat}\}_{\mathit{KEY}} \qquad \textit{encrypted SAML assertion}$$

It is important to note that this *encrypted SAML assertion* conforms to the SAML standard [24].

In the notation introduced in the previous section, the scenario of federated registration can be specified as follows:

$$\begin{array}{llll}
c_0 & U \mapsto \mathit{IdP} & : & r \\
c_1 & \mathit{IdP} \mapsto \mathit{SP} & : & \{r, \mathit{SAML\,assertion}\}_{K_{\mathit{IdP}}^{-1}} \\
c_2 & \mathit{SP} \mapsto U & : & \{ok/ko\}_{K_{\mathit{SP}}^{-1}}
\end{array}$$

First, *IdP* intercepts U's request r to *SP* in which U also asks *IdP* and *SP* to federate. Second, *IdP* forwards r to *SP*, together with a *SAML assertion* proving that U was authenticated by *IdP* and containing the token. The *SAML assertion* is encrypted to preserve secrecy of the token and the whole message is signed by *IdP* to guarantee authenticity. Third, upon having examined the *SAML assertion*, *SP* grants/denies U access through a message signed by *SP*.

Next we present a specification of this federated registration scenario in Crypto-CCS. This specification is more expressive than the one in standard notation given above, because all operations and security checks on the various messages are explicitly modelled. Each process is parameterized by the terms or variables it has in its knowledge (from the beginning or because it received them earlier).

$$\begin{array}{ll}
\multicolumn{2}{l}{U_0(r) \doteq} \\
\quad c_0!r. & \textit{send request,} \\
\quad c_2?x_{\mathit{sign}}. & \textit{receive signature,} \\
\quad [x_{\mathit{sign}} \quad K_{\mathit{SP}} \vdash_{\mathit{ver}} x_{\mathit{acc}}].\mathbf{0} & \textit{verify signature and stop}
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{IdP}_0(0, n_U^{\mathit{IdP}}, id_U) \doteq} \\
\quad c_0?x_r. & \textit{receive request and} \\
\quad \mathit{IdP}_1(x_r, n_U^{\mathit{IdP}}, id_U) & \textit{go to next state}
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{IdP}_1(x_r, n_U^{\mathit{IdP}}, id_U) \doteq} \\
\quad [id_U \quad \mathit{auth} \vdash_{\mathit{pair}} (id_U, \mathit{auth})] & \textit{create pair,} \\
\quad [(id_U, \mathit{auth}) \quad n_U^{\mathit{IdP}} \vdash_{\mathit{pair}} ((id_U, \mathit{auth}), n_U^{\mathit{IdP}})] & \textit{create pair,} \\
\quad [((id_U, \mathit{auth}), n_U^{\mathit{IdP}}) \quad \mathit{KEY} \vdash_{\mathit{enc}} & \\
\qquad \{((id_U, \mathit{auth}), n_U^{\mathit{IdP}})\}_{\mathit{KEY}}] & \textit{encrypt pair,} \\
\quad [x_r, \{((id_U, \mathit{auth}), n_U^{\mathit{IdP}})\}_{\mathit{KEY}} \quad k_{\mathit{IdP}}^{-1} \vdash_{\mathit{sign}} x_{\mathit{sign}}] & \textit{sign pair,} \\
\quad c_1!x_{\mathit{sign}}.\mathbf{0} & \textit{send SAML assertion + request and stop}
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{SP}_0(0) \doteq} \\
\quad c_1?x_m. & \textit{receive SAML assertion + request} \\
\quad \mathit{SP}_1(x_m) & \textit{and go to next state}
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{SP}_1(x_m) \doteq} \\
\quad [x_m \quad k_{\mathit{IdP}} \vdash_{\mathit{ver}} x_p] & \textit{verify signature,} \\
\quad [x_p \vdash_{\mathit{2nd}} x_{\mathit{enc}}] & \textit{extract encryption,} \\
\quad [x_{\mathit{enc}} \quad \mathit{KEY} \vdash_{\mathit{dec}} x_{\mathit{dec}}] & \textit{decrypt,} \\
\quad [x_{\mathit{dec}} \vdash_{\mathit{1st}} x_{\mathit{pair}}] & \textit{extract pair: token + Auth Stat,} \\
\quad [x_{\mathit{dec}} \vdash_{\mathit{2nd}} x_{n_U^{\mathit{IdP}}}] & \textit{extract nonce,} \\
\quad [x_{\mathit{pair}} \vdash_{\mathit{1st}} x_{id_U}] & \textit{extract token,} \\
\quad [x_{\mathit{pair}} \vdash_{\mathit{2nd}} x_{\mathit{auth}}] & \textit{extract Auth Stat,} \\
\quad [x_{\mathit{auth}} \vdash_{\mathit{check}} x_{\mathit{auth}}] & \textit{test correctness Auth Stat,} \\
\quad [x_{n_U^{\mathit{IdP}}} \vdash_{\mathit{check}} x_{n_U^{\mathit{IdP}}}] & \textit{test freshness nonce,} \\
\quad [x_{id_U} \quad x_{n_U^{\mathit{IdP}}} \vdash_{\mathit{pair}} (x_{id_U}, x_{n_U^{\mathit{IdP}}})] & \textit{build pair to store,} \\
\quad c_S!(x_{id_U}, x_{n_U^{\mathit{IdP}}}) & \textit{store token + nonce pair,} \\
\quad [\mathit{access} \quad k_{\mathit{SP}}^{-1} \vdash_{\mathit{sign}} x_{\mathit{sign}}] & \textit{prepare signature to} \\
\quad c_2!x_{\mathit{sign}}.\mathbf{0} & \textit{grant access and stop}
\end{array}$$

For the sake of readability, we did not fully spell out the digital signatures (*i.e.* we just applied the private key to the message to be signed). Moreover, we assumed a direct communication channel between *SP* and U in order to grant/deny access, while in reality all communication passes through *IdP*.

The federated registration process *FR* is described by the parallel composition $\mathit{FR} \doteq U_0 \parallel \mathit{IdP}_0 \parallel \mathit{SP}_0$.

B. Federated Network Providers

The second scenario that we formalize involves two federated network providers: a fixed operator *FO* and a mobile operator *MO*. Our starting point is when U initiates the process of federated registration with *SP* through *MO*. From [21] we inherit the assumption that all communication between *FO* and *MO* is secure: we consider them to share a secret key $\mathit{KEY}_{\mathit{FM}}$.

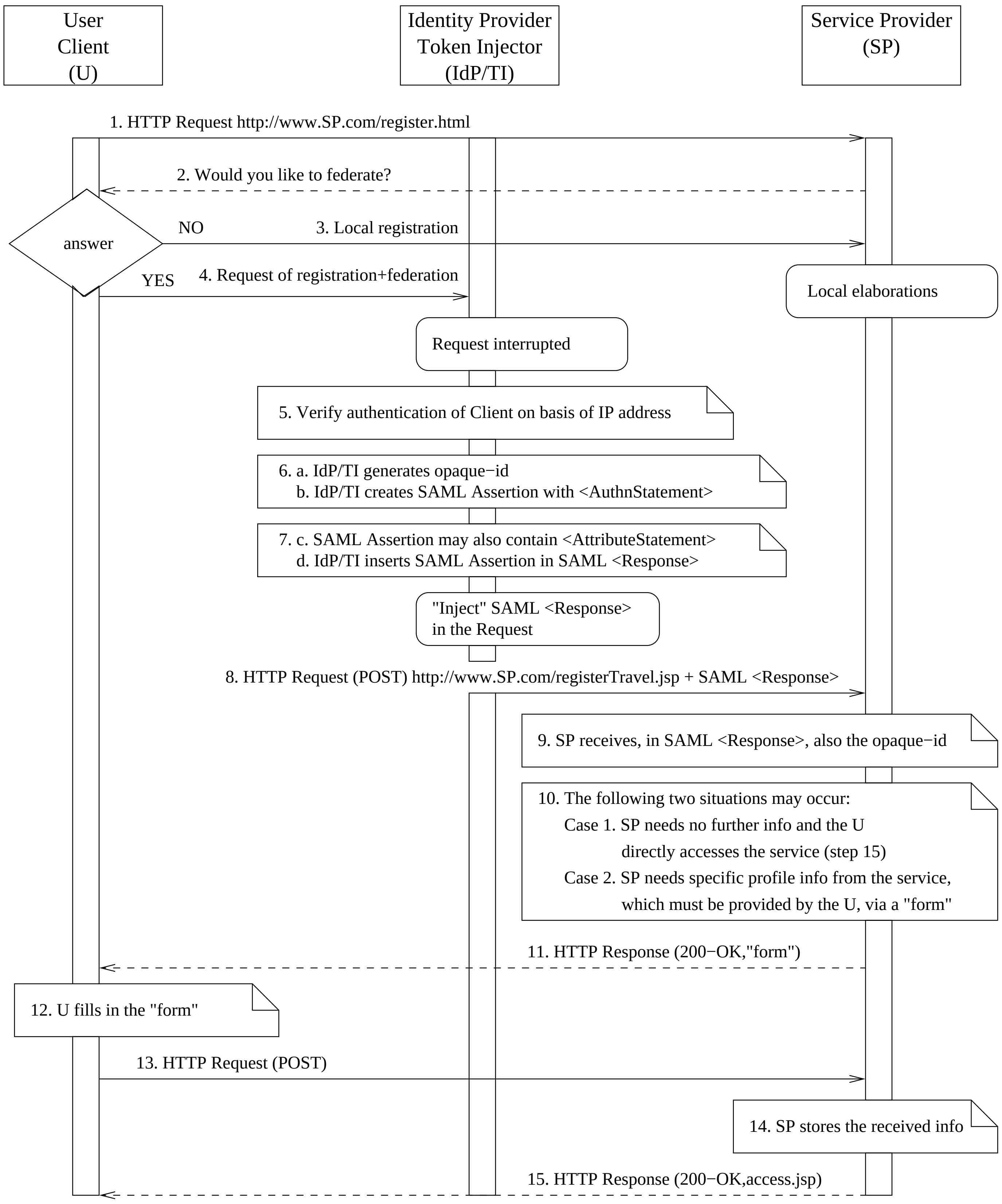

User Client (U)
Identity Provider Token Injector (IdP/TI)
Service Provider (SP)
1. HTTP Request http://www.SP.com/register.html
2. Would you like to federate?
answer
NO
3. Local registration
YES
4. Request of registration+federation
Local elaborations
Request interrupted
5. Verify authentication of Client on basis of IP address
6. a. IdP/TI generates opaque−id
b. IdP/TI creates SAML Assertion with <AuthnStatement>
7. c. SAML Assertion may also contain <AttributeStatement>
d. IdP/TI inserts SAML Assertion in SAML <Response>
"Inject" SAML <Response> in the Request
8. HTTP Request (POST) http://www.SP.com/registerTravel.jsp + SAML <Response>
9. SP receives, in SAML <Response>, also the opaque−id
10. The following two situations may occur:
Case 1. SP needs no further info and the U directly accesses the service (step 15)
Case 2. SP needs specific profile info from the service, which must be provided by the U, via a "form"
11. HTTP Response (200−OK,"form")
12. U fills in the "form"
13. HTTP Request (POST)
14. SP stores the received info
15. HTTP Response (200−OK,access.jsp)

Fig. 4. Message sequence chart of federated registration.

Furthermore, to avoid dealing twice with the same process of token generation, we propose a formalization that slightly enriches the procedure presented in [21], [23]: as soon as one of the two network providers receives a request from U, it searches its repository for a token already associated to U. If this token is found, then it is retrieved and the procedure continues as in the federated registration scenario (generating a new nonce each time a *SAML assertion* is sent to *SP*). If, on the other hand, this token is not found, then it is generated and immediately sent to the other federated network provider, where the token is stored for subsequent interactions between *U* and *SP*, after which the procedure continues its usual course:

$$\begin{array}{lll} c_0 & U \mapsto MO & : \; r \\ c_{MF} & MO \mapsto FO & : \; \{id_U, U\}_{KEY_{FM}} \\ c_1 & MO \mapsto SP & : \; \{r, SAML\,assertion\}_{K^{-1}_{MO}} \\ c_2 & SP \mapsto U & : \; \{ok/ko\}_{K^{-1}_{SP}} \end{array}$$

The Crypto-CCS specifications of processes *U* and *SP* are as in the first scenario. Next we present the specifications for the processes *MO* and *FO*.

$$\begin{array}{lr} MO_0(0, n_U^{MO}, id_U, KEY_{FM}) \doteq & \\ \quad c_0?x_r. & \textit{receive request and} \\ \quad MO_1(x_r, n_U^{MO}, id_U, KEY_{FM}) & \textit{go to next state} \end{array}$$

$$\begin{array}{lr} MO_1(x_r, n_U^{MO}, id_U, KEY_{FM}) \doteq & \\ \quad [id_U \quad U \vdash_{pair} (id_U, U)] & \textit{create pair,} \\ \quad [(id_U, U) \quad KEY_{FM} \vdash_{enc} \{(id_U, U)\}_{KEY_{FM}}] & \textit{encrypt pair,} \\ \quad c_{MF}!\{(id_U, U)\}_{KEY_{FM}}. & \textit{send token to FO,} \\ \quad [id_U \quad auth \vdash_{pair} (id_U, auth)] & \textit{create pair,} \\ \quad [(id_U, auth) \quad n_U^{MO} \vdash_{pair} ((id_U, auth), n_U^{MO})] & \textit{create pair,} \\ \quad [((id_U, auth), n_U^{MO}) \quad KEY \vdash_{enc} & \\ \qquad \{((id_U, auth), n_U^{MO})\}_{KEY}] & \textit{encrypt pair,} \\ \quad [x_r, \{((id_U, auth), n_U^{MO})\}_{KEY} \quad k_{MO}^{-1} \vdash_{sign} x_{sign}] & \textit{sign pair,} \\ \quad c_1!x_{sign}.\mathbf{0} & \textit{send SAML assertion + request and stop} \end{array}$$

$$\begin{array}{lr} FO_0(KEY_{FM}) \doteq & \\ \quad c_{MF}?x_{enc} & \textit{receive encryption,} \\ \quad [x_{enc} \quad KEY_{FM} \vdash_{dec} x_{dec}] & \textit{retrieve decryption,} \\ \quad c_S!x_{dec}.\mathbf{0} & \textit{store token + identity pair and stop} \end{array}$$

The whole process is described by $U_0 \parallel MO_0 \parallel FO_0 \parallel SP_0$.

C. Multiple Service Providers

The third and final scenario that we formalize involves two service providers. To access their services, U issues two separate requests. Hence *IdP* must insert two different nonces and two different tokens in the two *SAML assertion*s:

$$\begin{array}{lll} c_1 & U \mapsto IdP & : \; r_{SP_1} \\ c_2 & IdP \mapsto SP_1 & : \; \{r_{SP_1}, SAML\,assertion^{r_{SP_1}}\}_{K^{-1}_{IdP}} \\ c_3 & U \mapsto IdP & : \; r_{SP_2} \\ c_{acc_1} & SP_1 \mapsto U & : \; \{ok/ko\}_{K^{-1}_{SP_1}} \\ c_4 & IdP \mapsto SP_2 & : \; \{r_{SP_2}, SAML\,assertion^{r_{SP_2}}\}_{K^{-1}_{IdP}} \\ c_{acc_2} & SP_2 \mapsto U & : \; \{ok/ko\}_{K^{-1}_{SP_2}} \end{array}$$

This specification obviously reflects only one particular order of arrival and management of the two requests. Due to lack of space, we omit the Crypto-CCS formalization of this scenario.

V. Security Properties and a First Analysis

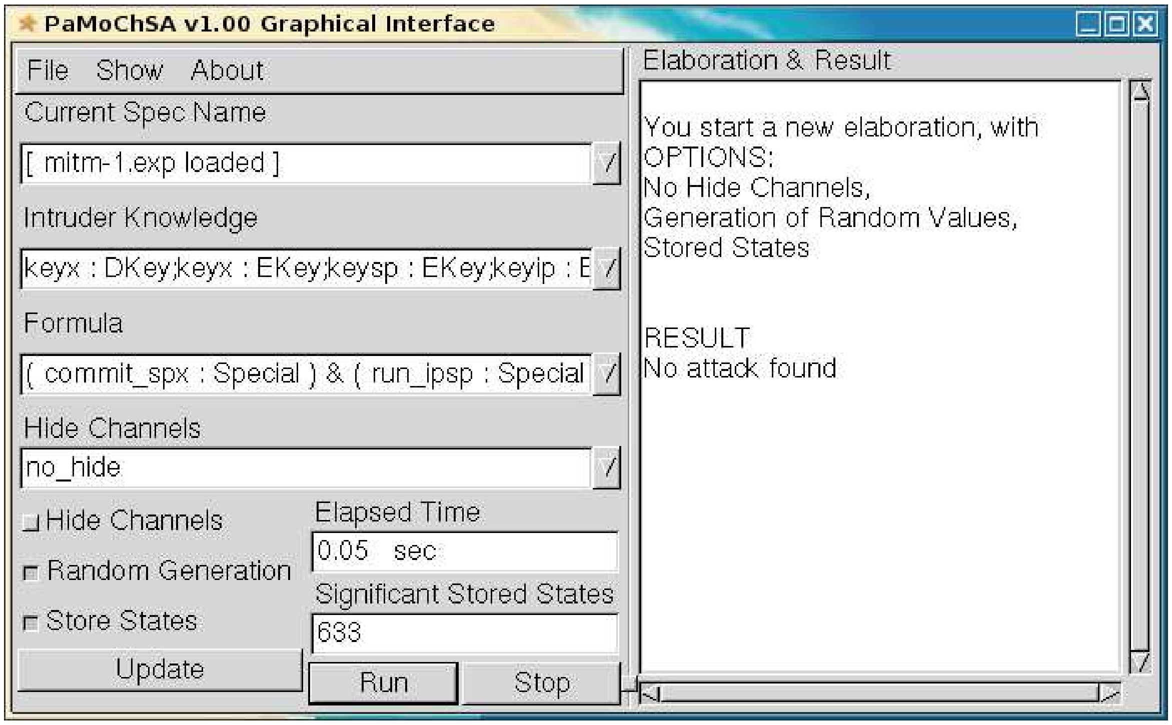
At the introduction of the identity federation protocol in [21], some rough guidelines on how to assure a minimal number of security properties were sketched. For one, it is noted that the token must be properly protected by means of cryptographic tools, like digital signatures, symmetric encryption and nonces, before it is sent from the network provider to the SP. The reason is obvious: this communication channel cannot be considered secure, since it is well beyond the network provider's security domain.

In the previous section we formally specified three different user scenarios of this network protocol. To define the structure of the exchanged messages, we were forced to consider all relevant security aspects in detail. This led us to use digital signatures to assure the messages' authenticity (*i.e.* to ensure that they were sent by who claims to have sent them) and integrity (*i.e.* to ensure that they were not modified on their route from sender to receiver), encryption to preserve the tokens' secrecy (*i.e.* to ensure that they cannot be revealed by unauthorized persons) and nonces to block messages from being replied (to avoid replay attacks). The user scenarios of the network protocol that we formalized in the previous section obviously have many more security aspects of interest.

As a first step towards a full-blown security analysis of the network protocol, we will verify below the vulnerability of the first scenario w.r.t. a man-in-the-middle attack. A man-in-the-middle attack is an adversary's attempt to intercept and modify messages between two trusted participants, in such a way that neither participant is able to find out that their communication channel has been compromised. We use model checking to perform the analysis. Model checking is an automatic technique to verify whether or not a system design satisfies its specifications and certain desired properties [29]. Such a verification is moreover exhaustive, *i.e.* all possible input combinations and states are taken into account. The level of completeness of a verification of course depends on the range of properties that are verified. Compared to testing, model checking generally needs to be performed on an abstract system (specification) to avoid state-space explosions. However, more problems are usually found by model checking the full behaviour of a scaled-down system than by testing some behaviour of the full system.

A. Analysis of a Man-in-the-middle Attack

In this section we verify whether or not the specification of the insecure channel between *IdP* and *SP* can withstand a man-in-the-middle attack, *i.e.* an adversary trying to intercept a conversation between *IdP* and *SP*. This boils down to verifying the following property: whenever *SP* concludes the network protocol apparently with *IdP*, it was indeed *IdP* that executed the protocol. To do so, we introduce two special actions in our Crypto-CCS specification: *commit*(*SP*,*IdP*) and



Current Spec Name
[mitm-1.exp loaded]
Intruder Knowledge
keyx : DKey;keyx : EKey;keysp : EKey;keyip : E
Formula
(commit_spx : Special) & (run_ipsp : Special
Hide Channels
no_hide
Hide Channels
Random Generation
Store States
Elapsed Time
0.05 sec
Significant Stored States
633
Elaboration & Result

```
You start a new elaboration, with
OPTIONS:
No Hide Channels,
Generation of Random Values,
Stored States


RESULT
No attack found
```

Fig. 5. Screenshot of PaMoChSA's graphical interface.

run(*IdP*,*SP*). The former action represents the fact that *SP* has indeed terminated the protocol with *IdP*, while the latter action represents the fact that *IdP* indeed started communicating with *SP*. The property is then translated into requiring action *run*(*IdP*,*SP*) to always precede action *commit*(*SP*,*IdP*).

We used the model checker PaMoChSA v1.0 [20] to verify this property. PaMoChSA requires the following input:

- a file with the protocol specification in Crypto-CCS;
- a logic formula expressing the property to be verified;
- the adversary's initial knowledge.

We considered an adversary X and set its initial knowledge to the set of public messages that it knows at the start of the protocol, *i.e.* the public keys of *IdP* and *SP* and its own public and private key denoted by pk_X and pk_X^{-1}. Consequently, the input and result of the analysis we performed are as follows:

- Specification file: `mitm-1.exp`
- Logic formula: ((*run*(*IdP*,*SP*) AND *commit*(*SP*,*X*)) OR ((*run*(*IdP*,*X*) AND *commit*(*SP*,*IdP*))
- Initial knowledge: $\{pk_X, pk_X^{-1}, pk_{IdP}, pk_{SP}\}$
- Result: **No attack found**

Figure 5 shows the graphical interface of the tool, with the loaded experiment and the result.

To verify the logic formula specified above, the tool set out to find a run of the protocol with the following characteristic: at the end of the run, the adversary knows either message (*run*(*IdP*,*SP*) AND *commit*(*SP*,*X*)) (*i.e. IdP* is convinced to have talked with *SP*, while in reality it was *SP* that has finished talking with X) or message (*run*(*IdP*,*X*) AND *commit*(*SP*,*IdP*)) (*i.e. SP* is convinced to have talked with *IdP*, while in reality it was *IdP* that has started talking with X). It took the tool less than a second to conclude that such a run does not exist. Hence the network protocol is correct w.r.t. the analyzed security property, *i.e.* it does not fall to a man-in-the-middle attack.

VI. Conclusion

The result of the security analysis presented in the previous section strengthens our confidence in the formal specifications of the three user scenarios that we presented in this paper. In particular, it leads us to believe that we correctly inserted digital signatures, encryption and nonces into the network protocol. This is a clear advantage of the use of formal methods in the design phase of a protocol: it allows one to eventually arrive at a well-defined protocol that is guaranteed to satisfy certain desirable properties. In the future we intend to extend our security analysis of the identity federation protocol by considering more user scenarios and all of the aforementioned security properties, *i.e.* authenticity, integrity, secrecy and absence of replay attacks.

Finally, an important component of security analysis has to do with performance and real-time issues. In the future we therefore hope to carry out experiments with quantitative extensions of formal methods and tools, *e.g.* timed, probabilistic and stochastic formal specification languages and stochastic model checkers. To this aim, we will first need to collect detailed statistical information on the typical use of the identity federation protocol in practice.

Acknowledgment

This research was supported by the EU project Sensoria.

References

[1] M. Abadi and A. D. Gordon, “Reasoning about Cryptographic Protocols in the Spi Calculus,” in *Proc. CONCUR’97*, ser. LNCS, vol. 1243. Springer, 1997, pp. 59–73.

[2] R. Focardi, R. Gorrieri, and F. Martinelli, “Non Interference for the Analysis of Cryptographic Protocols,” in *Proc. ICALP’00*, ser. LNCS, vol. 1853. Springer, 2000, pp. 354–372.

[3] D. Marchignoli and F. Martinelli, “Automatic Verification of Cryptographic Protocols through Compositional Analysis Techniques,” in *Proc. TACAS’99*, ser. LNCS, vol. 1579. Springer, 1999, pp. 148–162.

[4] C. Meadows, “Formal Verification of Cryptographic Protocols: a Survey,” in *Proc. ASIACRYPT’94*, ser. LNCS, vol. 917. Springer, 1995, pp. 135–150.

[5] G. Lowe and A. W. Roscoe, “Using CSP to Detect Errors in the TMN Protocol,” *Software Engineering*, vol. 23, no. 10, pp. 659–669, 1997.

[6] V. Shmatikov and U. Stern, “Efficient Finite State Analysis for Large Security Protocols,” in *Proc. CSFW’98*. IEEE Press, 1998, pp. 105–116.

[7] F. J. Thayer, J. C. Herzog, and J. D. Guttman, “Strand spaces: proving security protocols correct,” *Journal of Computer Security*, vol. 7, no. 1, pp. 191–230, 1999.

[8] G. Lowe, “Breaking and fixing the Needham-Schroeder public-key protocol using FDR,” in *Proc. TACAS’96*, ser. LNCS, vol. 1055. Springer, 1996, pp. 147–166.

[9] J. C. Mitchell, M. Mitchell, and U. Stern, “Automated Analysis of Cryptographic Protocols using Murphi,” in *Proc. S&P’97*. IEEE Press, 1997, pp. 141–153.

[10] A. W. Roscoe and M. H. Goldsmith, “The Perfect Spy for Model-checking Crypto-protocols,” in *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[11] F. Martinelli, M. Petrocchi, and A. Vaccarelli, “Formal analysis of some secure procedures for certificate delivery,” *Software, Testing, Verification and Reliability*, vol. 16, no. 1, pp. 33–59, 2006.

[12] M. Abadi and M. R. Tuttle, “A Semantics for a Logic of Authentication,” in *Proc. SPDC’91*. ACM Press, 1991, pp. 201–216.

[13] D. Kindred and J. M. Wing, “Fast automatic checking of security protocols,” in *Proc. 2nd Usenix Workshop on Electronic Commerce*, 1996, pp. 41–52.

[14] L. C. Paulson, “Proving Properties of Security Protocols by Induction,” in *Proc. CSFW’97*. IEEE Press, 1997, pp. 70–83.

[15] M. Abadi, “Secrecy by Typing in Security Protocols,” *Journal of the ACM*, vol. 46, no. 5, pp. 749–786, 1999.

[16] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson, “Static Analysis for the pi-Calculus with Applications to Security,” *Information and Computation*, vol. 168, no. 1, pp. 68–92, 2001.

[17] M. H. ter Beek, G. Lenzini, and M. Petrocchi, “A Team Automaton Scenario for the Analysis of Security Properties in Communication Protocols,” *Journal of Automata, Languages and Combinatorics*, accepted for publication, 2006.

[18] R. Focardi and F. Martinelli, “A uniform approach for the definition of security properties,” in *Proc. FM’99*, ser. LNCS, vol. 1708. Springer, 1999, pp. 794–813.

[19] F. Martinelli, “Analysis of security protocols as open systems,” *Theoretical Computer Science*, vol. 290, no. 1, pp. 1057–1106, 2003.

[20] Partial Model Checking Security Analyzer PaMoChSA v1.0. [Online]. Available: http://www.iit.cnr.it/staff/fabio.martinelli/pamochsa.htm

[21] M. Bonifati, P. De Lutiis, C. Moiso, E. Morello, and L. Sarchi, “Identity Federation for Services in Convergent Networks,” in *Proc. ICIN’06*, 2006, pp. 109–114.

[22] Liberty Alliance Project. [Online]. Available: www.projectliberty.org

[23] T. Wason et al. (2005) Liberty ID-FF Architecture Overview v1.2. [Online]. Available: www.projectliberty.org/liberty/specifications__1

[24] OASIS Security Services TC. (2005) Security Assertion Markup Language SAML v2.0. [Online]. Available: www.oasis-open.org/specs/

[25] D. Dolev and A. Yao, “On the Security of Public Key Protocols,” *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[26] G. Lowe, “An attack on the Needham-Schroeder public key authentication protocol,” *Information Processing Letters*, vol. 56, no. 3, pp. 131–136, 1995.

[27] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[28] H. R. Andersen, “Partial Model Checking,” in *Proc. LICS’95*. IEEE Press, 1995, pp. 398–407.

[29] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.