

Proving a WS-Federation Passive Requestor Profile

Thomas Groß
IBM Research Division
Rüschlikon, Switzerland
tgr@zurich.ibm.com

Birgit Pfitzmann
IBM Research Division
Rüschlikon, Switzerland
bpf@zurich.ibm.com

ABSTRACT

Currently, influential industrial players are in the process of realizing identity federation, in particular the authentication of browser users across administrative domains. WS-Federation is a joint protocol framework for Web Services clients and browser clients. While browser-based federation protocols, including Microsoft Passport, OASIS SAML, and Liberty besides WS-Federation, are already widely deployed, their security is still unproven and has been challenged by several analyses. One reason is a lack of cryptographically precise protocol definitions, which impedes explicit design for security as well as proofs. Another reason is that the security properties depend on the browser and even on the browser user. We rigorously formalize a strict instantiation of the current WS-Federation Passive Requestor Interop profile and make explicit assumptions for its general use. On this basis, we prove that the protocol provides authenticity and secure channel establishment in a realistic trust scenario. This constitutes the first positive security result for a browser-based identity federation protocol.

Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General—Security and protection; K.4.4 [Computers and Society]: Electronic Commerce—Security

General Terms

Security, Theory, Standardization

Keywords

Web service security, identity federation, federated identity management, single signon, web browser, browser-based, WS-federation passive requestor profile, WSFPI, security proof of protocols, security

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM Workshop on Secure Web Services, October 29, 2004, Fairfax VA, USA.
Copyright 2004 ACM 1-58113-973-X/04/0010 ..\$5.00

1. INTRODUCTION

Major industrial players are currently striving for solutions in identity federation. This technology implements user authentication and identity management across administrative domains. Identity federation is most relevant in business-to-business applications and aims at reducing user management costs dramatically, e.g., the cost of password helpdesks and of user registration and, even more importantly, the deletion of users from systems. Hence industry developed several proposals for suitable message formats as well as multi-party authentication and attribute exchange protocols. They are being widely implemented in middleware such as access control and user management products.

Protocols only based upon a standard web browser play an important role in identity federation, in particular for multi-party authentication and attribute exchange. These browser-based protocols do not require the installation of special client software and, therefore, have no footprint on client systems. This is called zero-footprint property. They are cost-efficient to deploy and form an easy entry point into identity federation. Thus they are the spearhead of identity federation technology and are expected to be widely used in the near future. Microsoft Passport [5] was the first proposal in this area. Although the protocol details were not public, several vulnerabilities were identified. The most detailed analysis can be found in [12]. However, some problems stated there are inherent in browser-based protocols. The Security Assertion Markup Language (SAML) [23] was the first open standard. It defines authentication and attribute tokens usable for identity federation, as well as basic profiles (protocols in typical security terms) for using these tokens. Several problems were recently found in a SAML profile [8]. The Shibboleth project for university identity federation can be seen as a more complex SAML profile [4]. SAML was also the basis for the Liberty Alliance project [19]. A vulnerability was found in one of the original Liberty protocols, the enabled-client protocol [17]; this problem was removed in subsequent versions. WS-Federation [10] is part of the IBM/Microsoft web services security roadmap. It links the Web Services world and the browser world by defining a joint identity-federation basis for both client types. Special aspects for the browser case are defined as a Passive Requestor Profile [11].

All analyses available for browser-based identity federation protocols have in common that they only state negative results, i.e., vulnerabilities; there is no positive security analysis in the whole area. Individual standards proposals typically come

with security considerations; however, these are all sorted according to certain known threats and countermeasures taken. Today it is commonly acknowledged that vulnerability analyses give no security guarantee and that security proofs are desirable for all new cryptographic protocols. We will give the first proof of a browser-based federation protocol in the following. There were multiple obstacles to overcome. First, the protocols rely on a browser as one of the protocol participants. All prior protocols analyzed cryptographically are assumed to be carried out by specific protocol machines that do nothing but executing the protocol as specified (unless the machine is corrupted). Hence we need a model of necessary browser capabilities as well as assumptions on what a browser will *not* do. Secondly, due to the limited capabilities of browsers, the user at the browser is an active participant and certain assumptions must be made about the user as well. Thirdly, all the relevant protocols are defined in the form of standards, i.e., with precise message formats but less precise descriptions of the protocols themselves. We therefore have to provide a more rigorous definition, but nevertheless remain faithful to the actual standards proposal.

We chose the WS-Federation Passive Requestor Interop scenario [9] as the first identity federation protocol to prove. Originally intended for interoperability testing, this scenario offers a concrete instantiation of WS-Federation similar to profiles in other standards proposals and corresponding to a particular cryptographic protocol. It is a natural basis for initial deployments of WS-Federation, in particular in cases where third-party authentication of browser users is the main goal. We consider a strict instantiation of the WS-Federation Passive Requestor Interop profile, i.e., we regard discretionary security-relevant constraints (“should” or “recommended”) as mandatory and prescribe the use of secure channels. While these measures are not necessary in certain scenarios, we include them to get a general-purpose protocol. We also elaborate on issues like the choice of metadata, setup, and tests during the protocol. We call this version of the WS-Federation Passive Requestor Interop that we actually prove the WSFPI protocol. The property we prove is correct authentication of the browser user, including the possibility to further use a secure channel set up with this user during the protocol. Many of the techniques used here could be reused in the analysis of other browser-based protocols, in particular the submodule interfaces, the assumptions on browsers and users, and the security requirement and overall proof structure.

1.1 Related Work

As long as we only consider the authenticity of the user at the end of an identity-federation protocol, we are dealing with 3-party entity authentication. This was introduced by Needham and Schroeder [14]. There is a large body of literature on the tool-supported analysis of such protocols based on abstractions of cryptography, starting with [13]. As typical classical 3-party authentication needs no specific cryptographic tricks, it was not a favorite object of study in cryptography, and we are not aware of any cryptographic proof before recent first proofs of the Needham-Schroeder-Lowe protocol [24, 1]. Establishment of a secure channel by a 3-party protocol is typically handled by the exchange of a session key. This holds for practical protocol proposals such as Kerberos and public-key infrastructures as well as for cryptographic protocols and

for tool-supported protocol analysis. In particular, [2] introduced a rigorous cryptographic treatment for protocols analyzed hitherto only based on unproven abstractions. The channel-establishment technique of browser-based federated identity-management cannot be modularized as key exchange followed by key usage because a standard browser would not use a key established in this way. Instead, the technique is to establish a channel with unilateral authentication first (concretely SSL or TLS without client certificates) and to use additional information sent in this channel for third-party authentication of the so far anonymous user of this channel. We therefore need slightly different security requirements than usual, besides the novel need to model a browser and its user. Secure channels without mutual authentication were first treated in [22]. Federated identity-management proposals typically treat such channels as a blackbox submodule, and so will we.

Federated identity protocols can also be analyzed for privacy. A detailed but informal treatment can be found in [16]. Some of the submodule definitions we use here were already made in [18] to describe a research proposal of browser-based federation protocol with optimal privacy. However, no security analysis was made there based on these definitions.

Web services security protocols were first analyzed in [7], which considers an abstraction of WS Security and the validation of the authentication of requests and responses. A second step in this direction is the development of a semantics for WS Security authentication [3], which is done by extending the XML data model and embedding it into the π -calculus. This work is over abstractions from cryptography, and does not treat federation protocols yet, in particular no protocols using browsers.

1.2 Organization

The remainder of the paper is structured as follows: We present an overview of the WSFPI protocol in Section 2. We define the submodules used by the protocol in Section 3 and describe the protocol steps in detail in Section 4. In Section 5, we specify the trust scenario and prove the authenticity and correct channel establishment of the protocol. Section 6 concludes the paper.

2. OVERVIEW OF THE WSFPI PROTOCOL

We first give an overview of the interop scenario from [9] in the strict instantiation mentioned in Section 1 and including details about the submodules and their set-up. We call this the WSFPI protocol. Figure 1 shows the message flow of the WSFPI protocol when no error occurs. Browser B communicates on behalf of user U with two other principals, C and S . User U wants to sign-in at an identity consumer C (“resource” or “destination site” in other terminologies) using WSFPI. An identity supplier S authenticates U and confirms its identity to identity consumer C by means of a signed SAML assertion.

Steps 1 and 10 show that user U is assumed to browse at identity consumer C before the protocol (Step 1) and to get some application-level response after the protocol (Step 10). In [9] these arrows are drawn to a “WS resource” different from the “resource IP/STS” that executes the protocol and thus corresponds to our identity consumer. This reflects that the browser will typically be intercepted or redirected when

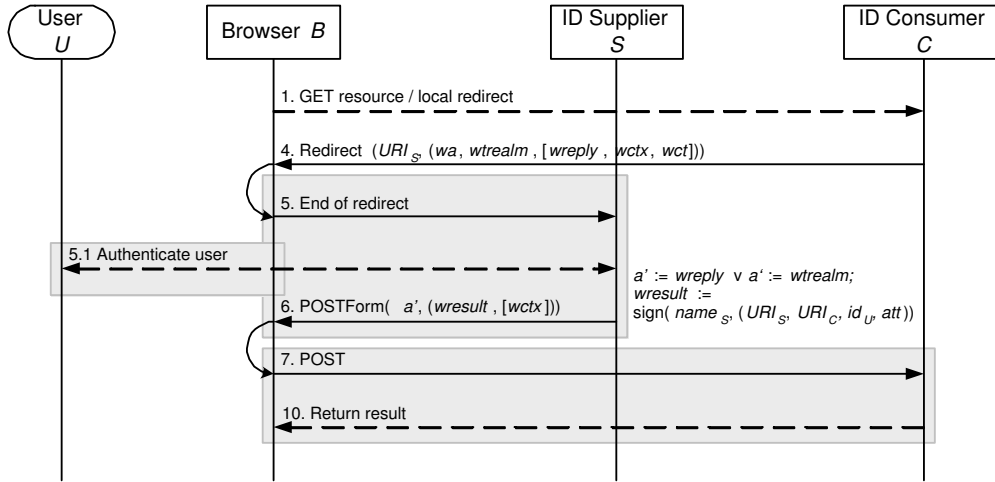


Figure 1: WSFPI protocol with abstract parameters. Steps with uninterrupted lines are actually specified in the protocol. The grey boxes denote secure channels.

accessing the resource. However, as [9] says nothing about Steps 1 and 10 except that they must be supported, this does not matter: we will in fact only analyze Steps 4 to 7. Steps 4-5 redirect the browser to identity supplier S , the unspecified Step 5.1 authenticates the user to the identity supplier, and Steps 6-7 essentially redirect the browser back to identity consumer C with a signed SAML assertion as response. The assertion contains an authentication statement and an attribute statement about user U .

The figure contains all the exchanged top-level parameters with their original names. In addition, the most important elements of $wresult$ are shown. For simplicity, we have omitted the time stamps in $wresult$ and corresponding verifications because they are not necessary for the authenticity property that we show. This may be added in the future. In both abstract messages, Redirect and POSTForm, the first parameter in the figure is the address and the second parameter the payload, here a list of the protocol parameters. Square brackets mean that parameters are optional. The end of a redirect message gets its parameters from the redirect message, and the POST message gets them from the payload of the message denoted POSTForm. In the latter case a form, typically including a script, is used to make the browser or user POST the message described.

3. REQUIRED SUBMODULES AND SET-UP

In this section, we review the submodules used by the WSFPI protocol. In addition, we define which data must be exchanged in advance for the submodules and the WSFPI protocol itself. We consider the parameters exchanged by the modules as metadata and summarize them in Table 1 at the end of this section.

3.1 Notation

We use a straight font for constants, including constant sets, functions, and submodules, and italics for *variables*. Assignment by possibly probabilistic functions is written as \leftarrow .

Assignment of a value to a tuple of variables means making correspondingly many projections; if one of these fails the entire assignment fails. We specify the communication with a submodule similar to [15, 18]; an input in to an asynchronous submodule $module$ is written $module!(in)$ and an output out from it $module?(out)$. Most of these submodules are distributed, i.e., they get inputs and make outputs for several participants of the main protocol. Simple message sending is shown as $-m \rightarrow$ or $\leftarrow m-$ between participants; concretely it stands for HTTP messages sent over an insecure channel.

We denote the set of URL host names by $URLHost$ and the set of URL host and path names by $URLHostPath$. We denote an address $a \in URLHostPath$ as a pair (h, p) of a host name $h \in URLHost$ and a path. The function $host$ denotes host extraction, i.e., $host(a) = h$ for $a = (h, p) \in URLHostPath$.

3.2 Browser Channels

Every identity supplier and every identity consumer must be able to maintain secure channels with standard browsers. In [9, p. 3f], this is described by HTTPS requirements for sending or receiving certain messages. We assume a particularly strict interpretation of these requirements, which basically enforces that the communication of Steps 5-10 is tunneled through unilaterally authenticated secure channels. We model such secure channels in the module $secchan$, which resembles SSL3.0 or TLS1.0 with server authentication. Note that some implementations of [9] might not implement the version we analyze here.

3.2.1 Submodule “secchan”

We denote the submodule for secure channels as $secchan$ and its possible actions as follows.

Browser	Server
1 $secchan!(new, adr)$	$\rightarrow secchan?(new, cid, adr)$
2 $secchan?(accept, cid, adr, id)$	$\leftarrow secchan!(accept, cid, id)$
3 $secchan!(send, cid, m)$	$\rightarrow secchan?(receive, cid, m)$
4 $secchan?(receive, cid, m')$	$\leftarrow secchan!(send, cid, m')$

Line 1 shows that the browser initiates a secure channel to an address $adr \in URLHost$. Recall that “!” denotes that this is

an input to an asynchronous module. The server is notified with a channel identifier cid . Line 2 shows that the server may accept the channel and identify itself under an identity $id \in \text{URLHost}$ (with $id = \epsilon$ for anonymity). The browser is notified of the acceptance and of id and cid . Then both parties may send messages. This is shown in Lines 3-4 with messages m and m' .

As the security of the channel, we assume that at most the partner from the channel initiation learns anything about messages sent as in Lines 3-4 (confidentiality) and that messages arriving as in Lines 3-4 have been sent in exactly this form by the partner (integrity). We also assume that the browser verifies $adr \subseteq id$ before outputting its acceptance in Line 2, where “ \subseteq ” denotes the standard address matching of HTTPS [21]. Essentially, this means that the chosen address adr is “covered by” or “under” the address id in the server certificate. Further aspects of the server authentication are described in Section 3.2.2.

3.2.2 Certificates Needed

Compared to [9], we fix in WSFPI how many keys per participating site are distributed and for what purposes. Here, we restrict WSFPI and model the secure channels as an independent module. Thus, we assume that its keys are not reused. Concretely, these keys have to come with SSL/TLS certificates acceptable to browsers. Abstractly, we require that each identity supplier S has an identity $ch_ids \in \text{URLHost}$ such that, if S uses $id := ch_ids$ in Line 2, a correct browser gets the output $\text{secchan?}(\text{accepted}, cid, adr, ch_ids)$. Nobody else must be able to achieve this, i.e., to impersonate the identity supplier under ch_ids as a secure-channel partner. More precisely, the channel will always be with an entity authorized by S to have such channels under ch_ids .

We make the same assumption for each identity consumer C with an identity $ch_id_C \in \text{URLHost}$.

3.2.3 User Involvement

We assume that browsers reliably present secure channels and the partner identity id to their users. This is implicit in the unspecified step 5.1 in WSFPI, but we need an assumption about the security of this step for the security analysis. The typical implementation is browser icons for windows with secure channels and the possibility to look up certificates. Identity suppliers may support this by personalizing the window content. We denote the outputs a user receives¹, and inputs that he makes, by

$$\begin{aligned} &\text{secchan?}(\text{receive}, id, m'); \\ &\text{secchan!}(\text{send}, id, m). \end{aligned}$$

We also assume that a browser user does not change during the lifetime of a secure channel.

3.2.4 Specific Abstract Browser Messages

We define abstract messages to model HTTP redirects and POST forms. The abstract message

$$\text{redirectS}(adr, path, query)$$

¹We only model that the user sees the partner identity id , not a channel identifier, because he or she will not notice if a channel is interrupted. Usually, however, a user can distinguish different channels with one partner by different windows.

models a redirect (HTTP 302 or 303) to

$$\text{https} : //adr/path?querystring,$$

where $querystring$ is an encoding of the abstract $query$. Its consequence is that the browser establishes a secure channel to the address adr and then sends $path$ and $querystring$ over that channel [21]. Similarly, $\text{POSTFormS}(adr, path, query)$ models a form containing a script that will POST a message whose body encodes the abstract $query$ to the address $\text{https} : //adr/path$.

3.3 Response Authentication

An identity supplier must authenticate SAML assertions in WSFPI. Generally, SAML allows arbitrary XML signatures [20], but WSFPI explicitly restricts this to public-key schemes and to use with X509 certificates (Pre-established Trust Relationship [9, p.11]). We write such signing as a function

$$m' \leftarrow \text{sign}(id, m)$$

for signing a message m under an identity id , and

$$(id, m) \leftarrow \text{test}(m')$$

for verifying a received message and extracting an identity id and a payload m . We denote failure by $(id, m) = (\epsilon, \epsilon)$. The functions include all necessary exchange and verification of keys and certificates.

In WSFPI, each identity supplier uses only one certificate for such signing. We call it $cert_S$ and the function that looks up the name from an X.509 certificate name_{X509} , and we abbreviate $\text{name}_S := \text{name}_{X509}(cert_S)$.² We do not prescribe whether $\text{name}_S = ch_ids$.

We assume that the underlying signature scheme is secure against adaptive chosen-message attacks [6], and that this property is lifted to the name-based version described here via secure certification, i.e., nobody except S can sign any new message m under name_S .

3.4 Submodule “uauth” and User Registration

We formalize the user authentication with a password over a secure channel by means of a submodule uauth on top of the secchan submodule. The goal is to identify the user for a specific secure channel with channel identifier cid . Let Users_S be a set that is held by an identity supplier S and denotes the user identities registered at S .

3.4.1 User Authentication Module “uauth”

The user and the identity supplier set up a method for later user authentication via a browser. We explicitly model passwords in order to analyze the security of the zero-footprint and browser-stateless case. Users who do not insist on these properties may set up higher-quality authentication, in particular with remote identity suppliers. The overall method must comprise means to protect the user from fake-identity-supplier attacks, because the browser arrives at the identity supplier (or an adversary) by redirection from an untrusted site. This means at least user education about verifying the identity-

²We simplified the handling of alternate names as it is not of central interest in this paper.

supplier certificate.³ We denote the submodule as `uauth` and the possible actions as follows:

	User		Identity supplier
1	<code>uauth?(start, ch_id_S)</code>	←	<code>uauth!(start, cid)</code>
2	<code>uauth!(do, ch_id_S, login)</code>	→	<code>uauth?(done, cid, id_U)</code>

Line 1 denotes that the identity supplier initializes user authentication for a secure channel with identifier `cid`. At the user, this leads to an output that asks for authentication. This output contains the identity supplier's identity `ch_idS` that the browser obtained in the set-up of channel `cid`. With current concrete implementations this happens automatically by the browser window. The user inputs login information `login` into the same window (Line 2), and the identity supplier derives a user identity `idU ∈ UsersS ∪ {ε}`, where `ε` denotes failure.

3.4.2 Exchanged Parameters

After registration of user `U` at identity supplier `S`, the user knows an identity `ch_idS` that the identity supplier can use for secure browser channels (compare Section 3.2), and the identity supplier knows an identity `idU` of the user. These identities of all registered users form the set `UsersS`. Further, they share login information `loginU,S`. We have to assume that the entropy of `loginU,S` is sufficiently large and the protocol `uauth` good enough so that, as long as `U` only uses correct browsers, an attacker cannot achieve that `S` obtains an output `uauth?(done, cid, idU)` for a channel `cid` of which `U` is not the channel partner. (There is at most one such partner by Sections 3.2.1 and 3.2.3.) Note that this assumption is not always fulfilled in practice; then every purely browser-based authentication protocol fails.

3.5 Attributes

If an identity supplier and an identity consumer interact, they need a common vocabulary for user attributes. We simplify this as sets `Attribute_Names` and `Attributes` of attribute names and name-value pairs. We denote the data store of user attributes in identity supplier `S` by `DBS` and the lookup function by

$$att \leftarrow \text{eval}(DB_S, id_U, att_n).$$

The inputs are the identity supplier's current data store `DBS`, the identity `idU` of a registered user, and a list `att_n ∈ Attribute_Names*`. The output is a list `att ∈ Attributes* ∪ {⊥}`, where `⊥`, spoken "undefined", stands for cases such as missing attributes.

In WSFPI, it is in addition assumed that every identity supplier `S` that will be called by an identity consumer `C` knows which attributes `C` wants. This is modeled by a variable `att_namesC`, and it is assumed that this variable is defined at least if `S = SupplierC` (see Section 3.6). WSFPI contains no privacy provisions or strength considerations. Those could probably be added very similar to [18].

3.6 Addresses

In this section, we name the addresses relevant for the protocol and briefly discuss the constraints that [11] and [9] impose on them.

³A dangerous feature in Passport and Liberty is "inline single signon", where the identity supplier uses a part of the identity consumer's window, because it disables such methods.

3.6.1 Identity Supplier Address

WSFPI contains no provisions for selecting a user's identity supplier; each identity consumer `C` is supposed to use only one identity supplier, which we denote by `SupplierC`.⁴ Each identity supplier `S` chooses an address `URIS ∈ URLHostPath` and makes it known to each identity consumer `C` with `SupplierC = S` in the set-up. We assume that `URIS` is covered by channel identity `ch_idS`, i.e., `host(URIS) ⊆ ch_idS`. Further we assume that `S` always carries out the WSFPI protocol at this address.

3.6.2 Identity Consumer Address and Security Realm

Each identity consumer `C` chooses an address `URIC ∈ URLHostPath` and makes it known to `SupplierC` in the set-up. We assume that `URIC` is covered by channel identity `ch_idC`, i.e., `host(URIC) ⊆ ch_idC`.

Given an assumption discussed in Section 3.6.3 that the element `wrealm` in requests equals `URIC`, we assume that `URIC` is (part of) a security realm for `C`. (This is a SHOULD in [11] for `wrealm`.) We interpret this as meaning that only services trusted by the principal `C` can obtain any certificate that covers any address `a ⊆ URIC`. With respect to our secure channels this means that if a correct browser obtains an output `secchan?(accepted, cid, host(a), id)` for such an address `a`, then the channel partner is `C`. In addition, we assume that if principal `C` hosts a service at such an address `a`, this service can handle the WSFPI protocol correctly and follows the same trust assumptions as `C`, and otherwise requests to `a` are mapped to the WSFPI base service at `URIC`.

3.6.3 Address Constraints for the Request Derivation

Requests have certain degrees of freedom, and we assume that an identity consumer `C` chooses the requests dynamically based on some context. The choice may be different for each `C`, hence we write it as `req ← make_requestC(ctxtC)`. This function for an honest participant `C` must guarantee that `req` is of the form `req = (wa, wrealm, wreply, wctx, wct)` with

- `wa = wsigin1.0`,
- `wrealm = URIC`,
- `wreply ∈ URLHostPath ∪ {⊥}` with `wreply ⊆ wrealm` if `wreply ≠ ⊥`,
- `wctx ∈ String ∪ {⊥}` where `String` denotes the XML string type,
- `wct ∈ Time ∪ {⊥}` where `Time` denotes a particular XML time format.

In all these cases, `⊥` means that the parameter is not present in the XML representation.

3.7 Summary of Metadata

We summarize the metadata from the various submodules in Table 1.

The rows denote who knows metadata, and the columns about whom these metadata are. The first element in each field

⁴Note that `SupplierC` is not explicit metadata, but a "real identity" like `C` and `S`.

↓Who/about→	U	S	C
U	-	*: $ch_id_S, login_{U,S}$	-
S	*: $id_U, login_{U,S}$	-	*: URI_C, att_n_C
C	-	1: $cert_S (\rightarrow name_S), URIS$	1: ch_id_C

Table 1: Summary of metadata

↓Who/about→	U	S	C
U	-	$MetaS: ch_id, login$	-
S	$MetaU: id, login$	-	$MetaC: URI, att_n$
C	-	$certS (\rightarrow nameS), URIS$	ch_id

Table 2: Summary of metadata in database notation

is the multiplicity of the given relationship. It shows that each identity consumer only knows one supplier in WSFPI, while all other relationships are unrestricted. The field C/C was used for metadata that C needs but that is not pre-distributed to other participants. Otherwise we tacitly assume that participants know their own metadata. The arrow notation for $name_S$ means that it can be derived from $cert_S$.

The indices U, S, C are not metadata for the protocol, but only for us in the analysis and definition. We assume that each participant stores the metadata of a single other participant together, but we do not assume it knows any “real identity” of that participant beyond the explicit metadata. For navigation, i.e., lookups of one parameter given another, we use a database representation for the *-relationships. The databases and attributes are shown in Table 2; note the systematic relation to the names in Table 1. An identity consumer C simply has variables for the metadata of its one identity supplier. No uniqueness requirements for any attributes are made in WSFPI, but we assume that ch_id in $MetaS$ is a key attribute, i.e., unique, and similarly id in $MetaU$ and URI in $MetaC$. This uniqueness assumption implies a strict security of the certification process and the user registration in the setup phase of WSFPI. We denote the selection of an attribute from a database entry by a dot, e.g., $e.att_n$ denotes the attribute att_n in entry e . Selection in a database D by a predicate $pred$ is written as $D[pred]$.

4. WSFPI STEP BY STEP

We now define the individual steps of WSFPI, including parameter generation and tests. In contrast to Figure 1, which showed the error-free case, we now use different variables for values that can be different under attack, e.g., $wctx$ at the identity consumer in Steps 4 and 7. We only do this per participant; variables of different participants are implicitly qualified with the participant name. In addition to the participants’ long-term parameters, let ctx_C denote the contexts of C in which the current WSFPI execution starts.

4.1 Redirect to Identity Supplier

In Steps 4-5, the identity consumer C redirects the browser to its identity supplier, i.e., to the address $URIS$ from its metadata. This is a secure redirect as defined in Section 3.2, in which the query string transports the request parameters. Note that the initial redirect message in Step 4 is sent through an insecure channel and that the secure channel is not established

until Step 5. Thus Step 4 works as depicted in Figure 2.

Figure 2 also shows that the browser reacts in Step 5 by establishing a secure channel and sending the path and query-string over it. We call the channel identifier cid_{bs} for browser-supplier channel id. Typically, a message with the given $path$ triggers a WSFPI execution at the identity supplier. As we only specify WSFPI here, not the dispatching, we only abort if the path is wrong.

The notation ids on the browser side of Line 5b indicates that this is an identity supplier identity, but not yet known to be that of a specific identity supplier S known by the address ch_id_S to U . In Step 5d, identity supplier S enforces the constraints on the request parameters as specified in Section 3.6. Note that we include the timing parameter wct only for completeness; it is not relevant for the authenticity proof.

4.2 Authenticate User

After successful execution of Step 5, the identity supplier authenticates a user over the established secure channel with channel identifier cid_{bs} . With the notation from Section 3.4, Step 5.1 is defined in Figure 3.

In Step 5.1a, ids is the identity that the browser obtained in Step 5b. The user checks whether this is the identity of one of its identity suppliers. If yes, the user inputs the corresponding login information. The identity supplier derives a registered identity idu or aborts.

4.3 Deriving a Response

If Step 5.1 was passed, the identity supplier tries to derive a response. As no explicit request and no privacy are considered here, this is quite simple: The identity supplier looks up which attribute names the supposed identity consumer wants, and looks up the corresponding attributes of the user in its database DB_S . We assume that it identifies the identity consumer C by the parameter $wrealm$ of the request because this is the only mandatory parameter related to C . Thus it can reuse the metadata entry e_C retrieved above.

The assignment to $wresult$ stands for producing a SAML assertion and wrapping it as a RequestSecurityTokenResponse as prescribed in WSFPI: The identity supplier signs certain parameters under its identity $name_S$ (and using the certificate $cert_S$). The main parameters signed are the identity supplier’s address $URIS$ as the SAML Issuer element, the identity consumer’s identity $e_C.URI = wrealm$ as the Audience element, the user identity as the Subject, and the attributes att . Recall that we

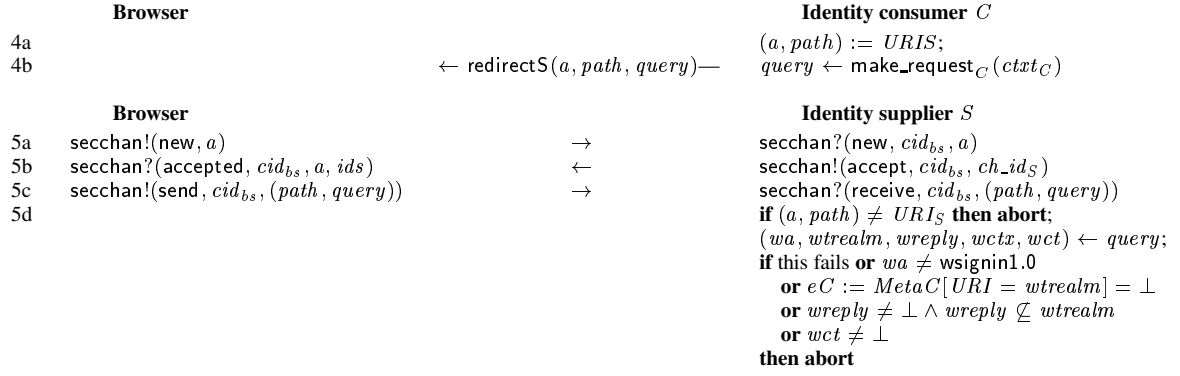


Figure 2: Redirect to identity supplier (Steps 4 and 5)

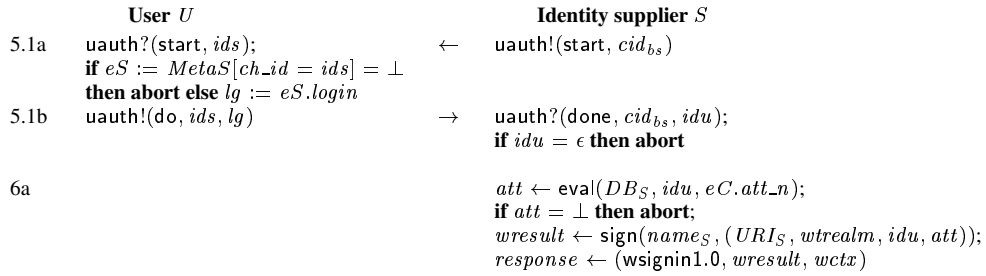


Figure 3: User authentication and response derivation (Steps 5.1 and 6)

have not represented time limits for simplicity because the basic authenticity property does not depend on them. Further, we have not represented how the subject and attributes are distributed over one or two SAML statements; this does not matter because the subject is unique in this representation in WSFPI. We have also not represented that an AppliesTo element outside the SAML assertion may repeat the information from Audience. The most important aspect is that all the parameters we represent as signed must indeed be signed in WSFPI.

4.4 POST Form Back

The identity supplier now essentially redirects the browser back to the identity consumer with the response $wresult$. However, this is done as a scripted POST, i.e., the submit command issuing the form with $wresult$ is fired by a script. The script, in turn, is sent within the secure channel with the identifier cid_{bs} . The return address is $wreply$ if this parameter was present in the (correct) request, else $wrealm$, and it must be an HTTPS address. We depict the details of this flow in Figure 4.

In Step 7d, the identity consumer tests the signature and validates that the issuer is its identity supplier as indicated by the name in its metadata. Then it retrieves the parameters from the payload; this corresponds to the validation of the SAML token and the RequestSecurityTokenResponse wrapper. Finally it verifies that the Audience element refers to itself. We represent the result of this protocol run as an output with the name `accepted` (similar to the outputs of the submodules), the channel identifier for which authentication was performed, and the

user identity idu and attributes att .

Typically C will now retrieve its context $ctxt_C$ by the parameter $wctx$ and internally redirect the browser to its original target URL, with the resulting identity and attributes translated into an internal format. The user can, to some extent, also keep track of the secure channel after the protocol by the identity ch_{id_C} that the browser obtains; recall Section 3.2.3.

5. SECURITY

We provide a property-based security proof for the WSFPI protocol specified in Section 4. The property we show is that at the end of a WSFPI protocol run, the identity consumer C has established a secure channel with a specific user U . This is more than entity authentication of user U , because the authentication is bound to the secure channel. Such binding is important to guarantee that a potential result in Step 10 of Figure 1 really goes to the identified user and that potential future requests in this channel come from the same user.

This authenticity only holds under additional assumptions. Besides typical trust assumptions about the honesty of certain participants (including correctness of their software), these are assumptions that guarantee the separation of the WSFPI protocol from other protocols. For most other cryptographic protocols, such separation assumptions are implicit in the definition of the protocol machines, which do nothing but what one defines for them, but for a protocol involving a browser and a user, we have to make these assumptions separately.

Definition 1. (Trust and Protocol Separation Assumptions) When we consider an identity consumer C and a potential user

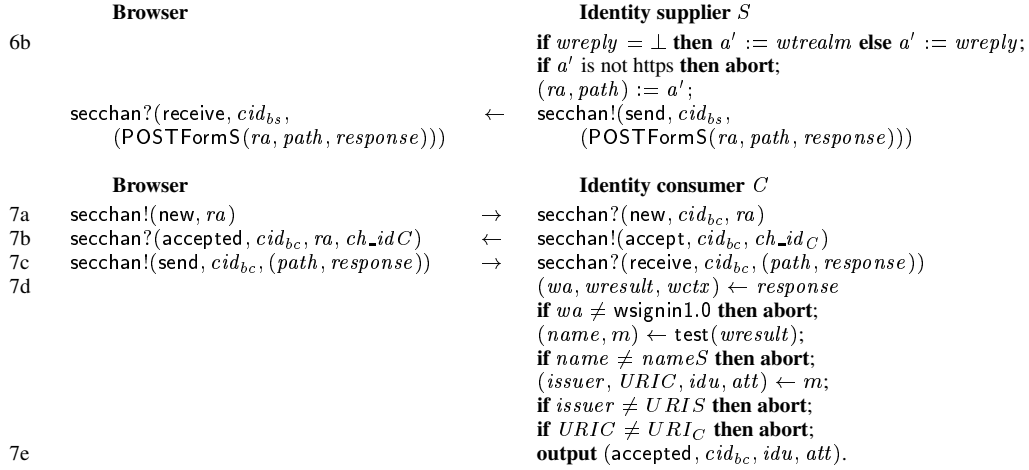


Figure 4: Scripted POST to the identity consumer (Steps 6 and 7)

identity idu , we make the following assumptions in addition to the assumptions already made about submodules:

- *Honest parties*: Identity consumer C , its identity supplier S , and the (at most one) user U that is registered under idu at S are honest, and all browsers that U used so far are correct.
- *Information flow*: None of these parties does anything with information received in a WSFPI protocol including setup except what the protocol and its submodules prescribe, and with the exception that C can use the final output from Step 7e.
- *Cross-protocol key use*: The overall use of the secret key referred to in $cert_S$ excludes cross-protocol attacks, i.e., no other application uses this key to sign messages that pass the verifications in Step 7d from the third line onwards.

The information flow assumption is certainly not trivial for the browsers, as they are protocol-unaware and follow their own rules for the transactions of WSFPI. The cross-protocol key use assumption is most easily realized if the secret key corresponding to $cert_S$ is only used for signing SAML tokens for the WSFPI protocol, not even for other WS-Federation subprofiles. Thus both assumptions need careful evaluation in practice.

THEOREM 1 (AUTHENTICITY OF WSFPI). *Let the assumptions of Definition 1 for an identity consumer C and a string idu be true and let all the submodules of WSFPI fulfill the corresponding security assumptions. Then the following holds: If the identity consumer C obtains an output (accepted, cid_{bc}, idu, att) from the WSFPI protocol, then the secure channel with channel identifier cid_{bc} is a channel with the only user U that has the identity idu at C 's identity supplier S .*

PROOF. Our proof is structured as follows: We first show that C only gets the final positive output from the WSFPI protocol if its channel partner has a token $wresult$ from C 's identity supplier S , meant for C and designating the identity as

idu . Then we show that only U , its browser B , and C can get such a token from S . Here we exploit the provisions for secure channels and correct addressing as well as the cross-protocol key use assumption besides the protocol definition. Finally, the trust and information flow assumptions imply that U , B , and C will not pass such a token to others; hence U must indeed be C 's current channel partner. We now show this in detail.

If C obtains the output (accepted, cid_{bc}, idu, att) from the WSFPI protocol, then this output is made in Step 7e. This implies that C obtained a message $secchan?(receive, cid_{bc}, (path, response))$ where $response$ passed the tests of Step 7d. The assignment $(name, m) \leftarrow test(wresult)$ and the verification that $name = nameS$ imply that $name \neq \perp$ and thus the signature test was successful, and $nameS$ is by definition of the metadata the name of the identity supplier of C , which we now call S . Hence, by the assumed adaptive chosen-message security of the signature scheme, lifted to names, at some point in the past S signed the message m . By the absence of cross-protocol attacks, this must have happened in an execution of the WSFPI protocol by S . Recalling the assumption that the secure channels use separate keys, this can only have happened when signing a SAML assertion in the third line of Step 6a.

Now we investigate where S might have sent the token $wresult$ when it signed it. (We do not know yet whether this is the same protocol run as where C receives the token.) A precondition for Step 6a is that Step 5.1 was passed, i.e., S has a secure channel to the user mentioned in the token. More precisely, S has obtained a message $uauth?(done, cid_{bs}, idu)$ with a registered user identity idu and some channel identifier cid_{bs} . Let U be our index for the user with this identity at S ; there is only one by the assumptions about metadata in Section 3.7, and it has only used correct browsers so far by the trust assumption. Hence the assumption on $uauth$ implies that U is indeed the partner for the channel with identifier cid_{bs} . This same user identity idu is what S puts into the token in Step 6a.

The only action that S performs with the token $wresult$

is to send it in the same secure channel with identifier cid_{bs} in Step 6b. By the assumptions on secure channels in Section 3.2.1, only one partner can obtain this message, and we just saw that this must be the current browser B of user U .

By the information flow assumptions, neither B nor U does anything with this token except what is explicitly prescribed in the protocol. This is defined by the message (`POSTFormS(ra, path, response)`) that B may obtain in Step 6b. The assumptions on secure channels in Section 3.2.1 imply that this message arrives completely or not at all. Here S has constructed the POST address as $(ra, path) := a'$, where a' is one of $wreply$ and $wrealm$, which S derived in Step 5d. The tests in Step 6b imply that this is an https address. Hence POSTing to this address builds up a secure channel to address ra as described in Step 7a.

We show next that the partner for this channel is our identity consumer C . The tests in Step 5d imply that $a' \subseteq wrealm$. The identity supplier S uses the same value $wrealm$ as the second payload parameter in the signed token in Step 6a, i.e., as the Audience element in the SAML realization. As C accepted the token $wresult$ with this element, C 's verification in Step 7d implies $wrealm = URI_C$. Thus $a' \subseteq URI_C$. Hence if the browser accepts the channel establishment in Step 7b (and subsequently forwards the token $wresult$), then the channel partner is C by the security realm assumption of Section 3.6.2.

Furthermore, by the security realm assumption the service of C at the address $a' \subseteq URI_C$ handles WSFPI correctly and follows the information flow assumption about C . Hence C does not forward or reuse this token either, as it is not part of the output that C may reuse. Hence C 's receiving of the token in the protocol run we originally considered must be a direct consequence of the form posting from S via B , where U is the current user. Hence the channel with identifier cid_{bc} is indeed with user U . This finishes the proof. \square

6. CONCLUSION

We have proven the security of the Web Services Passive Requestor Federation Interop Scenario, here called WSFPI, based upon a rigorous definition of the profile in the style of cryptographic protocols. Our analysis points out assumptions needed for the security of WSFPI in a general deployment beyond the original interop test scenario. Our proof is the first positive security analysis for a browser-based identity federation protocol. As future work, we intend to elaborate on some of the underlying assumptions by making more detailed models of the submodules, in particular of browsers and users and the use of passwords. The need to model browsers and users was a primary new factor when proving a browser-based identity federation protocol, compared with proofs of other authentication protocols. We believe that the assumptions that we had to make about the submodules, while not trivial to fulfill in practice, are not specific to WSFPI, but largely shared by browser-based identity federation protocols.

Acknowledgements

We thank many colleagues in IBM for discussions about federated identity over time, and specifically Christopher Gibling, Daniel Lutz, and Michael Waidner for comments related to this proof.

7. REFERENCES

- [1] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003.
- [2] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *Proc. 27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 57–66, 1995.
- [3] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 198–209. ACM Press, 2004.
- [4] Scott Cantor and Marlena Erdos. Shibboleth-architecture draft v05, May 2002. <http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v0%5.pdf>.
- [5] Microsoft Corporation. .NET Passport documentation, in particular Technical Overview, and SDK 2.1 Documentation (started 1999), September 2001.
- [6] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
- [7] Andrew D. Gordon and Riccardo Pucella. Validating a web service security abstraction by typing. In *Proc. 2002 ACM Workshop on XML Security*, pages 18–29, Fairfax VA, USA, November 2002.
- [8] Thomas Gross. Security analysis of the SAML Single Sign-on Browser/Artifact profile. In *Proc. 19th Annual Computer Security Applications Conference*. IEEE, December 2003.
- [9] Matt Hur, Ryan D. Johnson, Ari Medvinsky, Yordan Rouskov, Jeff Spellman, Shane Weeden, and Anthony Nadalin. Passive Requestor Federation Interop Scenario, Version 0.4, February 2004. <ftp://www6.software.ibm.com/software/developer/library/ws-fpscenario2.d%oc>.
- [10] Chris Kaler and Anthony Nadalin (ed.). Web Services Federation Language (WS-Federation), Version 1.0, July 2003. BEA and IBM and Microsoft and RSA Security and VeriSign, <http://www-106.ibm.com/developerworks/webservices/library/ws-fed/>.
- [11] Chris Kaler and Anthony Nadalin (ed.). WS-Federation: Passive Requestor Profile, Version 1.0, July 2003. BEA and IBM and Microsoft and RSA Security and VeriSign, <http://www-106.ibm.com/developerworks/library/ws-fedpass/>.
- [12] David P. Kormann and Aviel D. Rubin. Risks of the Passport single signon protocol. *Computer Networks*, 33(1–6):51–58, June 2000.
- [13] Jonathan K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141,

- 1984.
- [14] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
 - [15] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 184–200, Oakland, CA, May 2001. IEEE Computer Society Press.
 - [16] Birgit Pfitzmann and Michael Waidner. Privacy in browser-based attribute exchange. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 52–62, Washington, USA, November 2002.
 - [17] Birgit Pfitzmann and Michael Waidner. Analysis of Liberty single-signon with enabled clients. *IEEE Internet Computing*, 7(6):38–44, 2003.
 - [18] Birgit Pfitzmann and Michael Waidner. Federated identity-management protocols — where user authentication protocols may go. In *Security Protocols—11th International Workshop*, Lecture Notes in Computer Science, Cambridge, UK, April 2003. Springer-Verlag, Berlin Germany. To appear, 2004.
 - [19] Liberty Alliance Project. Liberty Phase 2 final specifications, November 2003. <http://www.projectliberty.org/>.
 - [20] W3C Recommendation. XML-Signature syntax and processing, February 2002. <http://www.w3.org/TR/xmlsig-core/>.
 - [21] E. Rescorla. Internet RFC 2818: HTTP over TLS, May 2000.
 - [22] Victor Shoup. On formal models for secure key exchange. Research Report RZ 3120 (#93166), IBM Research, April 1999. Version 4, November 1999, available from <http://www.shoup.net/papers/>.
 - [23] OASIS Standard. Security assertion markup language (SAML), November 2002.
 - [24] Bogdan Warinschi. A computational analysis of the Needham-Schroeder-(Lowe) protocol. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 248–262, 2003.