Federated Identity-Management Protocols
–Where User Authentication Protocols May Go–

Birgit Pfitzmann and Michael Waidner

IBM Zurich Research Lab
{bpf,wmi}@zurich.ibm.com

Abstract. For authentication, one answer to the workshop question "where have all the protocols gone?" is "into federated identity management". At least this is what many influential industrial players are currently striving for. The best-known examples are Microsoft Passport, the Liberty Alliance's proposals, and WS-Federation. While there have been many political discussions about Passport, in particular its privacy, and some technical studies of operational risks, there is almost no public literature about the actual protocols and their security.

We start with an overview of the driving factors in this space, the security properties desirable and achievable under the given design constraints, and the protocols proposed so far. We present a new protocol, BBAE, with better privacy and scalability, i.e., absence of single points of control, than prior proposals. We also discuss particular difficulties of rigorously treating a protocol that can be a profile in current standardization efforts.[1]

1 Introduction

While cryptography and security research has devised security protocols for a wide range of functionality, such as payment systems and fair exchange, the majority of security protocols used in real networks and products only aim at user authentication and secure channel establishment. Research has also produced many good protocols achieving these goals; they are even the main focus of the use of formal methods in security. Nevertheless, a surprising number of new such protocols are continuously being designed in practice, and typically not by adopting well-investigated proposals. While lack of knowledge and a marketing-driven desire for novelty certainly play a role, new applications truly have new requirements, and it seems to require researchers to extend the existing basic ideas appropriately to these new requirements.

Federated identity management is a case in point. The high-level goal of enterprises is to simplify user management in an increasingly dynamic world. In particular, they want to benefit from user registration done in other places for their own user management. The market demand is mostly for business-to-business scenarios, less than the much-discussed initial scenario of Microsoft Passport where one enterprise essentially authenticates the world population. Current demand is even mostly for simplifications within one enterprise, and the next steps are scenarios like access control for employees of supply-chain partners and customer-relationship management in a federation of travel agencies, airlines, and hotels.

---

[1] This paper reflects the view of the authors, which is not necessarily shared by IBM.

The main requirement on current federated identity-management protocols is to work if the user agent is only a standard browser. This is called *browser-based* or *zero-footprint*. The reason is that a large percentage of potential users has proved to be unwilling to download specific software. Further, the solutions should work without active content, because many users turn that off for security reasons. Optimally, they should even work without cookies for the same reason; however, not all solutions do, and it is definitely accepted to take advantage of cookies if a browser returns them. Enterprises even require that the solution can be *browser-stateless*. This means that the user uses multiple browsers on different machines, up to working only from Internet cafes. While the latter is not advisable for security, nobody is forced to use Internet cafes at least in developed countries, and we simply accept this requirement.

In research terms, this amounts to three-party authentication, possibly under pseudonyms, combined with an exchange of user attributes. We call the third party *wallet* and the recipient of the authentication and attributes *destination site*. However, no state-of-the-art protocol applies because they all assume special user agents. In particular, a user agent in three-party authentication typically uses a newly generated or received cryptographic key, while a standard browser will not do this. We have to work with browser redirects and other standard HTTP constructs only. More precisely, HTTP, HTML, and secure SSL/TLS channels triggered by HTTPS addresses are assumed. Normal authentication protocols are known to be very prone to design errors, and the browser constraint can only make things worse. Hence we believe that detailed security considerations for such protocols will become very important. Further, the transfer of additional attributes and the possibly quasi-automatic usage adds privacy as an important dimension.

In this paper, we start with an overview of the security and privacy properties desirable and achievable under the zero-footprint and browser-stateless constraints, and the protocols proposed so far. We then present a new protocol, BBAE, with better privacy and scalability, i.e., absence of single points of control, than prior proposals. Between this workshop and these proceedings, scalability has been addressed in similar ways in standards proposals, while similar privacy is still not achieved.

We designed BBAE to be suitable as a profile in existing standardization efforts such as SAML, Liberty or WS-Federation, as this is the best chance of adoption. While this paper contains a more precise and mathematical definition of BBAE, this point of view had two consequences: First, a few parameters are used in specific ways to fit a particular standard. Secondly, the definition is more modular than usual in research definitions of security protocols. All current standardization efforts aim for such modularity, e.g., independence of underlying transport protocols, cryptographic mechanisms, and policies, and even relative independence of concrete message formats and message flows, the so-called profiles. (We discuss this tendency a bit more in the outlook.) Concretely, we have specified this version of BBAE completely as a SAML profile, i.e., with SAML message formats and general constraints. This specification was prototyped by Stephen Levy in the IBM Privacy Services prototype (for an earlier version see [3]). At the time of this workshop, SAML was the only choice, containing the only message standard with general attributes. The only real restriction by this choice was in the inclusion of privacy policies, see below. In the meantime, we also specified BBAE

as a WS-Federation Passive Requestor profile with very minor changes together with Daniel Lutz, ETH Zurich, who also prototyped that version. The existence of these profile specifications should not be misunderstood as IBM input to standardization.

A particular reason to present profiles of standards like SAML more formally is that each participant has several names in such profiles, e.g., in the certificate in an underlying SSL channel, in the certificate for an XML signature, and in metadata; in text representations it is hard to say precisely which name is used where. Additionally, there can be several channel and protocol identifiers.

Two other challenges in precisely describing a browser-based protocol are independent of a relation to standards: One has to describe the user and the browser. We are not aware of prior protocol definitions that include user rules. Providing them is important for a later security analysis, because in a zero-footprint, browser-stateless protocol the user is an important protocol principal, and security can only be defined with respect to certain user actions and shown under certain assumptions about the user behavior. It is vital for implementations and documentation of such protocols to be aware of this semantics of the users actions. We even recommend to standardize the corresponding user syntax, i.e., the graphical user interface, to a large extent.

2   Existing Proposals and Design Goals

Figure 1 gives an overview of browser-based federated identity-management protocols. The first such protocol was Microsoft Passport. It is not published, but guessable from existing publications [11, 9]. The only open standard so far is SAML [17]. Shibboleth [18] is a more elaborated SAML application to a university federation. The Liberty Alliance, also building on SAML, makes public proposals without an open standardization process [10]. WS-Federation belongs to a larger web-services roadmap by IBM and Microsoft; its passive requestor profile is the browser-based case [20]. BBAE is our proposal of this paper; it precedes Liberty 2 and WS-Federation. WebSEAL is a product from the enterprise space [8], its 2002 version is shown for comparison with standardization efforts. The round icons are browser-based protocols. For comparison, the square icons show other protocol classes for web authentication and attribute exchange, but not zero-footprint and browser-stateless. "Form filler" denotes local wallet products like [7, 16, 12, 23] and more recently browser personalization. "Proxy wallets" denotes similar products working as remote proxies. As they see the user's entire web traffic, they are much worse for privacy than the protocol class discussed here where the wallets only participate in authentication and attribute exchange. "PKI" means public-key infrastructure, and "idemix" (Identity Mixer) is an IBM prototype of unlinkable credentials [5].

Vertically we show to what extent a single *control point* is inherent in the protocol design. We call the opposite *scalable* because a single control point may be acceptable in one enterprise or a small federation, but not beyond. Control points occur for different reasons in protocols: the wallet addressing may be fixed, at least per installation, as in Passport; all wallets may have to trust each other; there may be no room for certificate chains in messages; or there may be a symmetric-key distribution center that can impersonate all other parties. A common cookie domain, as proposed by Liberty, is at
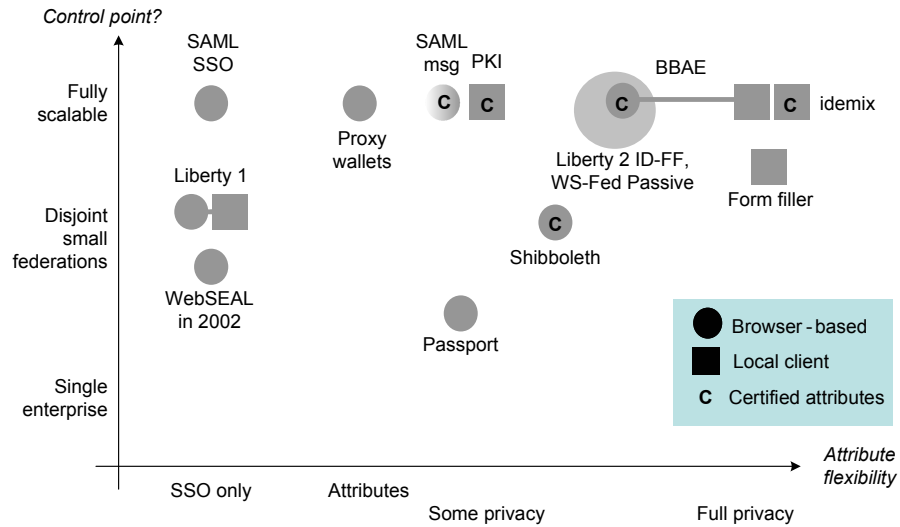
Control point?

SAML SSO

SAML msg PKI

BBAE

Fully scalable

C C

C idemix

Proxy wallets

Liberty 2 ID-FF, WS-Fed Passive

Form filler

Liberty 1

Disjoint small federations

C

WebSEAL in 2002

Shibboleth

Single enterprise

Passport

Browser - based

Local client

C Certified attributes

Attribute flexibility

SSO only   Attributes

Some privacy   Full privacy

Fig. 1. Overview of browser-based protocols and some related techniques.

least dangerous. However, we accept the lack of a fully specified set-up phase as long as such a phase could be added.

Horizontally, we first show whether only single signon is offered or multiple user *attributes* can be exchanged. We joined this with the *privacy* dimension because multiple attributes without privacy are simply a very long identity. Privacy on this protocol layer means that all information exchange can be governed by policy, and arbitrary policies are possible. In other words, no information is exchanged automatically. We include here that users are not forced to give attributes to third-party wallets, i.e., they can choose between the zero-footprint version and *local wallets* in the same protocol. One exception from privacy seems unavoidable for browser-based protocols: A wallet learns the identities of the destination sites that a user authenticates to. Detailed privacy goals and which protocol features enable them are discussed in [14]. Note that privacy in the overall system additionally requires a well-designed policy-management system and privacy on lower layers.

The "C" in some icons means that attributes may be certified, i.e., confirmed by the wallet or other parties. In most protocols certified attributes allow less privacy than uncertified ones (such as the user's book preferences). Combining certified attributes and strong anonymity is impossible with browser-based protocols. Where one needs that, one has to give up the zero-footprint restriction and upgrade to cryptographic unlinkable credentials [4], here exemplified by idemix [5].

Looking at browser-based protocols (round icons) only, and omitting the shaded "SAML msg" icon, which denotes that SAML already defines message formats for this case, but no protocols yet, one sees that no other protocol addresses full privacy, and no protocol prior to BBAE addressed attribute exchange for the fully federated case.

4

3 Overview of the BBAE Protocol

In this section, we present a browser-based attribute-exchange protocol, BBAE.

3.1 Message Flow

Figure 2 shows the message flow of the longest execution path of the BBAE protocol, and when no error occurs. Steps 0 and 13 show that the user is assumed to browse at the destination site before the protocol and to get some application-level response after the protocol. Steps 1-2 locate the wallet, Steps 3-4 redirect the browser to it, Step 5 authenticates the user to the wallet, Steps 6-10 transport the actual request and response on a backchannel, where Step 8 allows the real-time release of attributes, and Steps 11-12 redirect the browser back to the destination site with a handle that links this browser to the response. The figure contains all the exchanged parameters with hopefully self-
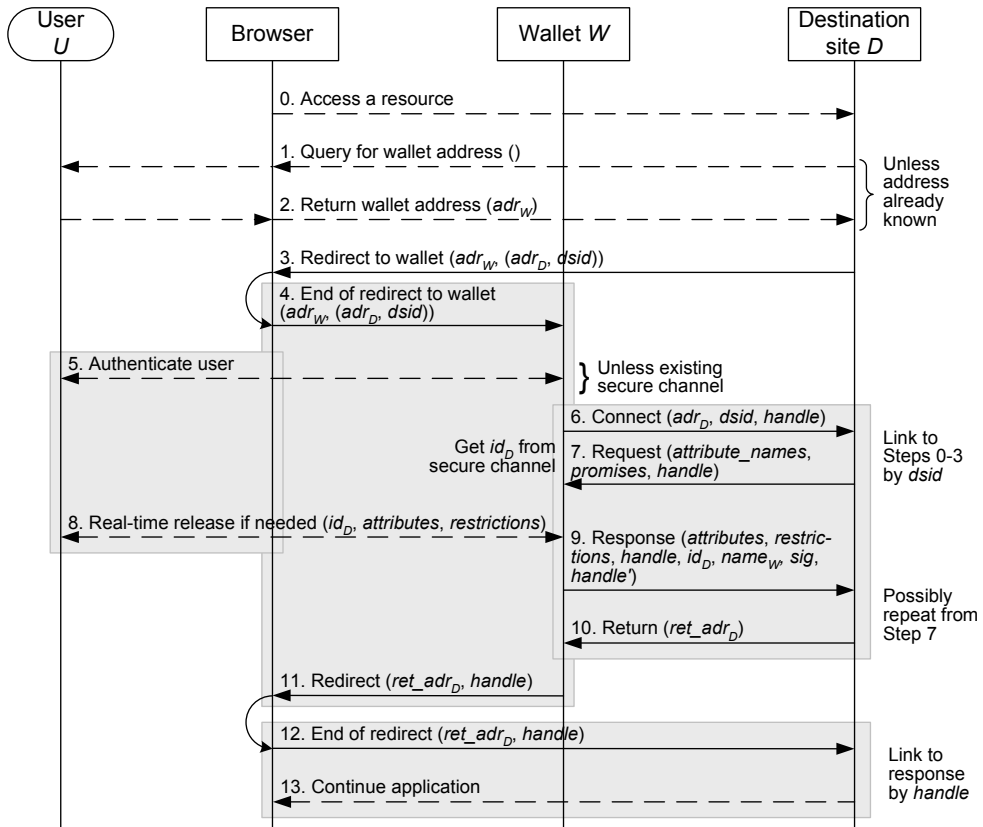

Fig. 2. BBAE protocol with abstract parameters. Steps with dashed lines are only needed in certain cases. The boxes denote secure channels.

explanatory names (*adr* denotes addresses and *dsid* a session identifier), except that all messages also contain a protocol identifier and a message number within the protocol. If not stated otherwise, these identifiers are not protected except by the secure channels.

The submodules used are defined in Section 4, the detailed protocol steps in Section 5, and Section 6 gives additional explanations when and why these steps are necessary.

3.2 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 ←. 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 sending of a message is shown as —*m* → or ← *m*— between participants; concretely it stands for HTTP messages.

4 Required Submodules and Set-up

Browser-based protocols, at least if one keeps close to standardization, use several submodules whose implementation is left open. Some submodules require that certain data is exchanged in advance; this corresponds to constraints on their global initial states. In this section, we describe these submodules and constraints. BBAE does not require parameter exchange except for its submodules, and in the concrete version this only means SSL server certificates and XML signature capabilities. This makes BBAE easier to bootstrap than all prior proposals. Negotiation of common submodules among multiple possible ones is not part of the subsequent protocol description. Further, we do not present details of how a user's authentication information, attributes, and privacy policies are managed over time.

4.1 Browser Channels

Every wallet and every destination site must be able to maintain secure channels with standard browsers. We define this to mean that message confidentiality and integrity hold with respect to the same principals for all messages sent over the channel. The easiest implementation is SSL/TLS channels, triggered by HTTPS addresses [21]. Servers that release SSL channels after each HTTP response may use another secure implementation, but should have at least one zero-footprint version.

Submodule "secchan". We denote the submodule as secchan and the possible actions as follows:

	Browser		Server
1	secchan!(new, *adr*)	→	secchan?(new, *cid*, *adr*)
2	secchan?(accepted, *cid*, *adr*, *id*)	←	secchan!(accept, *cid*, *id*)
3	secchan!(send, *cid*, *m*)	→	secchan?(receive, *cid*, *m*)
4	secchan?(receive, *cid*, *m'*)	←	secchan!(send, *cid*, *m'*)

Line 1 shows that the browser initiates a secure channel to an address $adr \in$ URLHost. This set denotes the set of URL host names, with localhost $\in$ 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. We denote anonymity by $id = \epsilon$. The browser is notified of the acceptance and of id and cid. Then both parties may send. This is shown in Lines 3-4 with messages m and m'.

The abstract message

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

models a redirect (HTTP 302 or 303) to https://$adr/path?querystring$, where $querystring$ is an encoding of the abstract $query$. Its consequences are that the browser establishes a secure channel to the address adr and then sends $path$ and $query$ over that channel [21]. Modeling the channel establishment within the redirect explicitly is important for a security analysis.

Certificates needed. In BBAE, wallets must be identified in secure browser sessions. Concretely, they need SSL/TLS certificates acceptable to browsers. Abstractly, we require that each wallet W has an identity id_W such that, if W uses $id := id_W$ in Line 2, a correct browser gets the output secchan?(accepted, cid, adr, id_W). Nobody else must be able to achieve this, i.e., to impersonate the wallet under id_W.

User involvement. Browsers must reliably present secure channels and the partner identity id to their users. The typical implementation is browser icons for windows with secure channels and the possibility to look up certificates. Wallets may support this by personalizing the window content. We denote the outputs a user receives, and inputs that he makes, by

$$\mathsf{secchan?}(\mathsf{receive}, id, m'); ^2$$

$$\mathsf{secchan!}(\mathsf{send}, id, m).$$

4.2 Backchannel Sessions

We also use secure channels between wallets and destination sites. We model these secure channels by the same module secchan as the browser channels, because they will typically also be SSL/TLS. However, as no party is a browser here, this implementation is easier to change. Again the initiating party need not identify itself; hence the in- and outputs are exactly as above. Destination sites must typically be identified; otherwise they only obtain the attributes that the users' privacy policies permit for anonymous recipients. Let id_D be the identity under which destination site D can securely identify itself in secure channels, in the same sense as for wallet identities id_W above. If a real organization has several such identities, we treat it as several destination sites because the identity choice can be made before a BBAE execution.

---

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

4.3 Response Authentication

A wallet must typically authenticate responses for a destination site. With SAML messages, this means XML signature capabilities [22]. They comprise both digital signatures in the cryptographic sense and authentication codes and allow many types of certification and key distribution. We discuss typical usages in Section 9. Generally, we write such authentication as a function

$$m' \leftarrow \mathsf{auth}(id, m, strength)$$

for authenticating a message m under an identity id with a certain strength, and

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

for verifying a received message and extracting an identity id, a payload m, and a strength. We denote failure by $(id, m) = (\epsilon, \epsilon)$ and the unauthenticated case by $strength = \epsilon$ and $m' = (id, m)$. We assume that a domain Strengths for the parameters $strength$ is given; for implementations without this notion we define Strengths to have one element only. The Liberty Alliance made detailed proposals [10]. The functions denote authentication including all necessary exchange and verification of keys and certificates.

4.4 User Registration

A user U must register with at least one wallet. Later, $Wallets_U$ denotes the set of wallets that U registered with, and $Users_W$ the set of user identities registered at a wallet W.

Installation or address. Let $local_W \in \mathsf{Bool}$ denote whether a wallet W is local. (This can be fixed for all users of W.) If W is local, it must be installed; this should be combined with a local firewall that only allows local access to this wallet. If W is remote, the user obtains a contact address $adr_W \in \mathsf{URLHost}$ of it (else let $adr_W := \epsilon$). This address must be fixed for all users of W unless a user consented to a different privacy policy. We assume a fixed path BBAEpath by which these addresses are extended to address the wallet services.

User authentication module "uauth". The user and the wallet 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 wallets. The overall method must comprise means to protect the user from fake-wallet attacks, because the browser arrives at the wallet (or an adversary) by redirection from an untrusted site. This means at least user education about verifying the wallet certificate.[3] We denote the submodule as uauth and the possible actions as follows:

---

[3] A dangerous feature in Passport and Liberty is "inline single signon", where the wallet uses a part of the destination site's window, because it disables such methods.

	User		Wallet
1	uauth?(start, id_W)	$\leftarrow$	uauth!(start, cid)
2	uauth!(do, id_W, $login$)	$\rightarrow$	uauth?(done, cid, id_U)

Line 1 denotes that the wallet initializes user authentication over a secure channel with identifier cid. Thus the module uauth uses the module secchan. At the user, this leads to an output that asks for authentication. This output contains the wallet's identity id_W 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 wallet derives a user identity $id_U \in Users_W \cup \{\epsilon\}$, where ϵ denotes failure.

Exchanged parameters. After registration of user U at wallet W, the user knows an identity id_W that the wallet can use for secure browser channels (compare Section 4.1), and the wallet knows an identity id_U of the user.[4] Further, they share login information $login_{U,W}$. We have to assume that the entropy of $login_{U,W}$ is large enough and the protocol uauth good enough that, as long as U only uses correct browsers, an attacker cannot achieve that W obtains an output uauth?(done, cid, id_U) for a channel cid where the attacker is the other principal. Note that this assumption is not always fulfilled in practice; then every browser-based protocol fails.

4.5 Attributes and Privacy Policies

Vocabulary. If a wallet and a destination site 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. Elements of Attribute_Names may contain further information, e.g., whether an attribute is mandatory in the answer or what degree of certainty or even liability of this attribute is needed. Similarly Attributes may contain such degrees of certainty. With SAML this would be easier if the attribute names could be of arbitrary types, not only strings. We assume a function auth_strength: Attributes* $\rightarrow$ Strengths that computes the necessary authentication strength for a list of attributes with their desired degree of certainty.

Privacy exchange language. Wallets and destination sites need a common privacy language because user attributes can typically only be forwarded for certain purposes and with certain obligations to the recipient. We assume fixed sets Promises and Restrictions for this. For simplicity, we let Promises comprise all additional information a destination site adds about itself, e.g., certified group memberships.

Privacy evaluation functions. Wallets need an internal privacy-management system (see, e.g., [3]). The languages Promises and Restrictions will typically be derived from the internal policy language by additional bindings to actual attribute names and by elements that characterize policies as promises or restrictions. The internal privacy policies

---

[4] If the name id_U is used for anything beyond recognition of U by W, it is considered an attribute and treated in Section 4.5, with examples in Section 9. Different roles that U wants to play with destination sites, but manage in one wallet, also belong to those sections. A person wishing to interact in different, unlinked roles with one wallet is treated as multiple users.

may be three-valued, i.e., the set of decisions is {allow, deny, askme}. Here allow and deny characterize situations where the user knows what he wants, while askme is for situations that the user does not want to think about before. Such an approach corresponds well to user studies that people do not want to start with abstract concepts like policies, but with examples. For the evaluation of privacy policies in a wallet, we therefore assume a function

$$(att, name_W, restr, rtr) \leftarrow \mathsf{priv\_eval}(DB, id_U, id_D, att\_n, prom).$$

The inputs are the wallet's current datastore DB including privacy policies, the identity id_U of a registered user, the identity id_D of a destination site, a list $att\_n \in$ Attribute_Names*, and promises $prom \in$ Promises by id_D. The outputs are

- a list $att \in$ (Attributes $\times$ {allow, askme})* containing the potentially permitted answers to att_n,
- a name $name_W$ under which the wallet should authenticate the attributes (see Section 9),
- their restrictions $restr \in$ Restrictions,
- and $rtr \in$ Bool denoting if a real-time release is necessary.

The evaluation function may have special properties like setting $att := ()$ if a requested mandatory attribute is denied.

Users also take privacy decisions, in particular for attributes where priv_eval outputs askme. To show what parameters the user needs and decides on, we write this as a function for each user U,

$$(att', name'_W, restr') \leftarrow \mathsf{priv\_eval}_U(id_D, att, name_W, restr^*, ctxt).$$

This defines that the user sees the identity id_D of the destination site, a proposed attribute list att and name $name_W$ as output by priv_eval with restrictions $restr^*$, and has a context $ctxt$ in which this data request occurs. For instance, U may release more attributes to D when buying something than when only browsing. The wallet should derive overall restrictions $restr^*$ as a suitable combination of the promises and the original restrictions,

$$restr^* \leftarrow \mathsf{priv\_combine}(restr, prom).$$

The user possibly modifies the attributes (in particular resolves the askme decisions), the name, and the restrictions.

Figure 3 shows how a wallet can present a privacy decision to the user. Where the wallet knows the attribute, it distinguishes the policy decisions allow (green, vertical stripes) and askme (red, diagonal stripes). In the example the name and shipping address are pre-authorized, while the ID number is not. The privacy restrictions are initially hidden under the button "Show privacy". The user might delete the ID number and not add an income because the fields are not mandatory. If they were, he might cancel. We define that registering for BBAE covers consent by U for W to use the BBAE protocol at all. The privacy policy must also govern all storage and usage of received information by W, but we need no notation for this because it concerns actions after the protocol.

Fig. 3. Simple example form for a user privacy decision (for the real-time release, Step 8).

Unless the policy allows otherwise all values that W receives in a BBAE execution must be deleted at the end of this execution.

Concrete user attributes and privacy policies. The user deposits attributes together with privacy policies, or consents to the use of already present attributes according to certain privacy policies. We denote the data set associated with user U in wallet W by $DB_{U,W}$.

Attribute verifications. If the wallet is to confirm certain attributes beyond just storing them for the user, it has to verify them. For names, this is discussed in Section 9.

4.6 Miscellaneous Submodules

Nonces. We also need nonce generation, written

$$n \leftarrow \mathsf{nonce\_gen},$$

where we omit the dependency on a security parameter k. It must have the usual cryptographic properties, and is implemented most easily as generation of a k-bit random string. SAML uses typed nonces (artifacts), but we do not distinguish such types here.

Destination site addresses. Each destination site D chooses two addresses $adr_D, ret\_adr_D \in \mathsf{URLHostPath}$. They must be fixed for all executions of the BBAE protocol, at least in a certain time period.

Session administration. For a destination site D, we need a set SID_D of current BBAE session identifiers, initially empty. It is important to define how D links the three parts of a BBAE execution that have no common channels from its point of view, Step 0-3, Step 6-10, and Steps 12-13. A similar set in wallets can remain implicit.

Location query form. Each destination site D needs a location query form

$$form \leftarrow \mathsf{loc\_form}_D(ctxt)$$

for Step 1, in case it has to ask a user in person for the address of his wallet. It can be a function of D's context. An example form is shown in Figure 4. The mandatory parts are at least one "no" or "cancel", a radio button $local \in \mathsf{Bool}$ where the text "it is local"

11

represents true and "my wallet holder is" represents false, and a text field adr after the latter button. These mandatory parts enable a consistent user experience over multiple destination sites and allow local proxies to handle location forms instead of the user. We represent the user's choices for the mandatory fields as a triple

$$(ok, local, adr) \in \mathsf{Bool} \times \mathsf{Bool} \times (\mathsf{URLHost} \cup \{\epsilon\}),$$

where $ok = \mathsf{true}$ if no "no" or "cancel" was chosen and the buttons $local$ were enabled, and $adr = \epsilon$ if the address field is empty or not in URLHost. The user should be able to verify whether a form $form$ has the mandatory parts, written as

$$ok' := \mathsf{verify\_form}(form)$$

with $ok' \in \mathsf{Bool}$. The user will decide whether he agrees to start BBAE and which wallet to use depending on his context. We write this

$$(ok, W) \leftarrow \mathsf{decideBBAE}_U(ctxt_U)$$

with $W \in Wallets_U \cup \{\epsilon\}$ and $(ok = \mathsf{false} \Rightarrow W = \epsilon)$.

<div style="border:1px solid black">

We would like to know something about you.
How can we find out?
○ No, I don't want to tell anything
○ I have a user name/password with you
○ I have a wallet and
  ○ ... it is local
  ○ ... my wallet holder is [＿＿＿＿＿]
[Submit]   [Cancel]
You can see who we are and our privacy policy.

</div>

Fig. 4. Example form for a wallet location query (Step 1)

Request derivation. For Step 7, the destination site D has to construct an attribute-name list $att\_n \in \mathsf{Attribute\_Names}^*$ that it asks for and promises $prom \in \mathsf{Promises}$ that it can make. This will be based on D's context, e.g., whether U is buying something from D or trying to access employee-only web pages. We write it as

$$(att\_n, prom) \leftarrow \mathsf{make\_request}_D(ctxt_D).$$

5 BBAE Step by Step

We now define the individual steps of BBAE, including parameter generation and tests. In contrast to Figure 2, which showed the error-free case, we now use different variables for values that can be different under attack, e.g., adr_W at the browser in Steps 2 and 4. We only do this per participant; variables of different participants are implicitly

qualified with the participant name. Further, some variables get shorter names than in the figure for readability in protocol tables.

In addition to the participants' long-term parameters, let $ctxt_D$ and $ctxt_U$ denote the contexts of D and U in which the current BBAE execution starts.

5.1 Find Appropriate Wallet

The wallet-finding phase, Steps 1-2, provides the destination site D with the host part adr_W of the wallet address for the subsequent redirection, where $adr_W = \mathsf{localhost}$ for local wallets. Possibilities for omitting this step are discussed in Section 6. The only general protocol version is that D asks the user for the wallet location. Thus these steps are (compare Section 4.6 for more explanations):

User U		Destination site D
1a		$dsid \leftarrow \mathsf{nonce\_gen};$
		$ctxt_{dsid} := ctxt_D$
1b **if** $\mathsf{verify\_form}(form) = \mathsf{false}$ **abort**;	$\leftarrow\!form\!\!-$	$form := \mathsf{loc\_form}_D(ctxt_{dsid})$
2a $(ok, W) \leftarrow \mathsf{decideBBAE}_U(ctxt_U);$		
2b $l := local_W; a := adr_W$	$-(ok, l, a) \rightarrow$	**if** $ok = \mathsf{false}$ **abort**;
		if $l = \mathsf{true}$
		then $adr_{dsid} := \mathsf{localhost}$
		else $adr_{dsid} := a$

In Step 1a, D generates a session identifier $dsid$ ("D's session id") for this execution of the BBAE protocol. It is used as an index for storing the context and the obtained address. The best way for D to link Steps 0 to 3 is to already use a secure channel, but we do not prescribe this because it only influences availability. Similarly, U is only sure about its context $ctxt_U$ if a secure channel is already used, but no serious damage can result if not.

5.2 Redirect to Wallet

In Steps 3-4, the destination site D redirects the browser to the obtained address. This is a secure redirect as defined in Section 4.1, and the address is extended with the fixed path from Section 4.4. The query string transports the destination site's backchannel address and session identifier. Thus Step 3 works as follows:

Browser		Destination site D
3a		$a' := adr_{dsid};$
		$path := \mathsf{BBAEpath};$
		$SID_D := SID_D \cup \{dsid\}$
3b	$\leftarrow \mathsf{redirectS}(a', path, query)-$	$query := (adr_D, dsid)$

The browser reacts by establishing a secure channel and sending the path and query-string over it. We call the channel identifier $bwid$ for browser-wallet id. A message with $path = \mathsf{BBAEpath}$ triggers a BBAE execution at the wallet. As we only specify BBAE here, not the dispatching, we only abort if the path is wrong. The wallet chooses a local session identifier $wsid$ ("W's session id") if BBAE really starts.

13

Browser		Wallet W
4a secchan!(new, a')	$\rightarrow$	secchan?(new, $bwid$, a')
4b secchan?(accepted, $bwid$, a', idw)	$\leftarrow$	secchan!(accept, $bwid$, id_W)
4c secchan!(send, $bwid$, $(path, query)$)	$\rightarrow$	secchan?(receive, $bwid$, $(path, query)$)
4d		**if** $path \neq$ BBAEpath **abort**;
		$wsid \leftarrow$ nonce_gen;
		$(a_{wsid}, dsid_{wsid}) := query$;
		if this fails **or** $a_{wsid} \notin$ URLHostPath **abort**

The notation idw indicates that this is a wallet identity, but not yet known to be that of a specific wallet W known as id_W to U.

5.3 Authenticate User

After successful execution of Step 4, the wallet authenticates a user over the established secure channel. With the notation from Section 4.4, Step 5 is defined as follows:

User U		Wallet W
5a uauth?(start, idw);	$\leftarrow$	uauth!(start, $bwid$)
if $idw \neq id_W$ **abort**		
5b uauth!(do, id_W, $login_{U,W}$)	$\rightarrow$	uauth?(done, $bwid$, idu_{wsid});
		if $idu_{wsid} = \epsilon$ **abort**

In Step 5a, idw is the identity that the browser obtained in Step 4b. The user remembers from Step 2a which wallet W he wanted to use and aborts if the secure channel is with an identity other than id_W.[5] In Step 5b, the user sends login information, and the wallet derives a registered identity or aborts.

5.4 Request

The wallet sets up a backchannel by contacting the destination site at the address obtained in Step 4d, and transmits the destination site's session identifier. In the concrete version, this is done in the query string of the HTTPS request that triggers the SSL/TLS channel. The wallet also transmits a fresh nonce $handle_{wsid}$ that is mainly needed to link Steps 9 and 12. This is due to a SAML idiosyncrasy. In SAML, this nonce is called an artifact.

Wallet W		Destination site D
6a secchan!(new, a_{wsid})	$\rightarrow$	secchan?(new, $wdid$, adr_D);
6b secchan?(accepted, $wdid$, a_{wsid}, idd_{wsid})	$\leftarrow$	secchan!(accept, $wdid$, id_D)
6c $handle_{wsid} \leftarrow$ nonce_gen		
6d secchan!(send, $wdid$, $(dsid_{wsid}, handle_{wsid})$)	$\rightarrow$	secchan!(receive, $wdid$, m);
		$(dsid, handle_{dsid}) := m$;
		if this fails **or** $dsid \notin SID_D$ **abort**

---

[5] Recall that unless we have a secure channel already, there is no unambiguous link to Step 2a for the user, but only availability is affected if the user assumes another wallet W' here.

The destination site D only reacts (at least with Step-6 behavior) if the address a_{wsid} is adr_D, as denoted implicitly in Step 6a; then it identifies itself under id_D. If a message then arrives over this channel, its first parameter must be the session identifier $dsid$ of a current BBAE execution. Then D can use $dsid$ to retrieve information from this protocol execution. It first derives what attributes it wants to ask for, and then sends the request together with the received nonce $handle_{dsid}$ over the secure channel.

	Wallet W		Destination site D
7a			$(att\_n, prom) \leftarrow \mathsf{make\_request}_D(ctxt_{dsid})$
7b	$\mathsf{secchan!}(\mathsf{receive}, wdid, (att\_n,$	$\leftarrow$	$\mathsf{secchan!}(\mathsf{send}, wdid, (att\_n,$
7b	$\quad prom, handle\_d))$		$prom, handle_{dsid}))$
7c	**if** $handle\_d \neq handle_{wsid}$ **abort**		

In the SAML version, the message in Step 7b is a SAML request containing exactly one SAML attribute query, and $handle_{dsid}$ is the artifact in that. The wallet aborts if this is not the handle it expects on the channel $wdid$. The wallet may now mark $handle_{wsid}$ as used so that Step 7c will abort if repeated as a backup security measure.

5.5 Deriving a Response

If Step 7c was passed, the wallet tries to derive a response. It first uses the privacy evaluation function from Section 4.5; it has all the necessary inputs in the current BBAE execution. Then, if necessary, it continues with a real-time release as explained in Section 4.5.

	User U		Wallet W
8a			$(att, name_W, restr, rtr) \leftarrow \mathsf{priv\_eval}(DB,$
			$idu_{wsid}, idd_{wsid}, att\_n, prom)$
8b			**if** rtr **then**
			$restr^* \leftarrow \mathsf{priv\_combine}(restr, prom);$
	$\mathsf{secchan?}(\mathsf{receive}, id_W, (idd, att,$	$\leftarrow$	$\mathsf{secchan!}(\mathsf{send}, bwid, (idd_{wsid}, att,$
	$name_W, restr^*))$		$name_W, restr^*))$
8c	$(att', name'_W, restr') \leftarrow \mathsf{priv\_eval}_U($		
	$idd, att, name_W, restr^*, ctxt_U)$		
8d	$\mathsf{secchan!}(\mathsf{send}, id_W, (att', name'_W,$	$\rightarrow$	$\mathsf{secchan?}(\mathsf{receive}, bwid, (att', name'_W,$
	$restr'))$		$restr'))$
			else $(att', name'_W, restr') := (att,$
			$name_W, restr)$

The wallet uses the attributes att' and restrictions $restr'$ as the main part of its response to the destination site. It adds the name idd_{wsid} of the destination site as a measure against man-in-the-middle attacks and the nonce $handle_{wsid}$. It authenticates these elements under $name'_W$ with the appropriate strength. (Recall Section 4.5 and that typical choices of this name are discussed in Section 9.) Similar to Step 6, we include a new fresh nonce $handle'_{wsid}$ in case Steps 7 and 9 will be repeated. The destination site makes the natural verifications. If it decides not to repeat Step 7, it tells the wallet the return address ret_adr_D for the browser.

	Wallet W		Destination site D

9a $str \leftarrow \mathsf{auth\_strength}(att')$;
  $sig \leftarrow \mathsf{auth}(name'_W, (att', restr',$
    $idd_{wsid}, handle_{wsid}), str)$;
  $handle'_{wsid} \leftarrow \mathsf{nonce\_gen}$

9b $\mathsf{secchan!}(send, wdid, (sig, handle'_{wsid})) \;\rightarrow\; \mathsf{secchan?}(receive, wdid, (sig, handle'_{dsid}))$

9c $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (name_W, m, str) \leftarrow \mathsf{test}(m')$;
  $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ **if** $(name_W, m) = (\epsilon, \epsilon)$ **abort**;
  $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (att, restr, idd, handle^*) := m$;
  $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ **if** this fails **or** $idd \neq id_D$
  $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ **or** $handle^* \neq handle_{dsid}$ **abort**;

10 $\mathsf{secchan?}(receive, wdid, ret\_adr) \;\leftarrow\; \mathsf{secchan!}(send, wdid, ret\_adr_D)$;
  $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad SID_D := SID_D \setminus \{dsid\}$

We did not formalize how D decides if it wants to repeat Step 7, e.g., because it is not satisfied with the attributes, and how $handle'$ then becomes $handle$. All other decisions that D bases on the attributes have no consequence in BBAE and are thus not formalized either, but D must delete attributes if it cannot fulfill the restrictions for them. The channel $wdid$ can now be released.

In the SAML version, Step 9 is a POST of a SAML response, where $handle_{wsid}$ is a SAML artifact included as SAML subject confirmation, idd_{wsid} is the SAML recipient element, $name'_W$ the issuer element, and the strength str is only implicit in the presence or absence of an XML signature and its type.

5.6 Redirect Back

If the wallet obtains a Step-10 message, it redirects the browser back to the destination site with $handle_{wsid}$ as a parameter. The redirect is sent within the secure channel with the identifier $bwid$, and the redirect must also trigger a secure channel.

	Browser		Wallet W

11 $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (ra, path) := ret\_adr$;
  $\mathsf{secchan?}(receive, bwid, \qquad\qquad \leftarrow \mathsf{secchan!}(send, bwid,$
    $(\mathsf{redirectS}(ra, path, handle_{wsid}))) \qquad\qquad (\mathsf{redirectS}(ra, path, handle_{wsid})))$

This has the following effect:

	Browser		Destination site D

12a $\mathsf{secchan!}(new, ra) \;\rightarrow\; \mathsf{secchan?}(new, bdid, ra)$
12b $\mathsf{secchan?}(accepted, bdid, ra, id_D) \;\leftarrow\; \mathsf{secchan!}(accept, bdid, id_D)$
12c $\mathsf{secchan!}(send, bdid, (path, handle_{wsid})) \;\rightarrow\; \mathsf{secchan?}(receive, bdid, (path, handleb))$
12d $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ **find** $dsid$ **with** $handle_{dsid} = handleb$

That D inputs id_D for this channel helps the user keep track of the channel after the BBAE protocol. D uses $handleb$ to retrieve the corresponding session identifier $dsid$ and thus also the context $ctxt_{dsid}$ and the response from Step 9 of the protocol execution with $dsid$. This finishes the BBAE protocol. Typically D will internally redirect the browser to its original target URL, stored in $ctxt_{dsid}$, with parameters from the response.

6 Optimizing the Number of Steps

In this section, we sketch why we believe the steps in BBAE are optimal for the general case, and in which cases steps can be omitted. As in the entire paper, we consider the situation where the user is initially browsing at the destination site. A portal scenario, where the user starts at the wallet and addresses destination sites via special links, only starts with Step 4.

The two redirects, Steps 3-4 and 11-12, are common to all known browser-based protocols. They seem unavoidable if the user and the browser are unknown at the destination site and the browser contains no user-related information.

Steps 1-2, i.e., asking the user for the wallet location, is optional in BBAE. It is necessary in the browser-stateless case, i.e., when neither the destination site nor the browser knows anything about the user. It is also needed if a user has several wallets without clear separation of responsibilities that could be evaluated automatically. Multiple unlinked wallets are, e.g., Microsoft's proposal for handling multiple roles; only in Passport they are all at the same address. With a standard browser and free choice of wallets, and without an overall cookie domain, the steps are also needed at least once per user and destination-site domain. For the other cases, implementers may choose any state-of-the-art techniques of state keeping, such as cookies or browser extensions. The steps then either disappear completely into Step 0, or do not go to the user.

Step 5 is clearly needed if no prior secure channel is available. Further, reusing an existing channel can be dangerous if only cookies are used for session maintenance and in the mobility scenario; compare attacks on Passport [19]. Step 5 cannot be joined into the real-time release (Step 8) if that becomes necessary, although one can postpone it until after Step 7: Before the real-time release, a remote wallet must know whose attributes it is looking up, and every wallet must authenticate its user before showing privacy-critical attributes on his or her screen for releasing.

The backchannel (Steps 6-10) can be omitted if both request and response are short enough to fit into a URL. As a protocol extension, the destination site could indicate this in Step 3, i.e., immediately add a request there and even omit the address adr_D if it does not expect a long response. However, if one adheres to the HTTP recommendation of at most 255 bytes, this does not often work except in small federations with a-priori exchanged symmetric keys, because signatures and certificates are too long. An alternative to backchannels for long data are POSTs. However, this requires user interaction between Step 3 and 4 and between Step 11 and 12, or active content for cross-domain posting, which is not zero-footprint. The inconvenience of user interaction can only be avoided by integrating it with something meaningful for the user. We do not see such a possibility for the first redirection. The second one can be integrated with the real-time release where that is present, but this greatly reduces flexibility, e.g., for multi-step interaction and for making privacy choices also for future similar requests. Hence we believe a backchannel is the better solution. It may also often be faster.

Steps 6 and 10 are needed because we build the backchannel from the wallet side. This is done to enable anonymous local wallets. Those can be addressed as "localhost" by the destination site in Steps 3-4, but the destination site cannot build a backchannel to them.

7   Multi-Wallet Extensions

Deriving a response between Steps 7 and 9 may involve a second wallet that holds or confirms certain attributes. We sketch the four main options. (No prior proposal contains confirmed attributes.) If the second wallet has to confirm a long-term attribute for a certain name, e.g., the year of birth, it can give the first wallet a confirmation token, e.g., a SAML attribute assertion, with an appropriate validity period. This becomes an attribute in the first wallet. If the confirmation must be fresh and the second wallet's policy allows release to the first wallet, the first wallet can fetch the confirmation on a backchannel. For a real-time release at the second wallet, the first wallet must redirect the browser to it. If the wallets are not linked, the first wallet only answers what it can and the destination site has to restart the protocol, asking the user for the appropriate wallet for the remaining attributes.

8   Security

The main issue with a security protocol should be security. Nevertheless, all prior protocols for federated identity management come with almost no published security justification. (The most detailed one is that of SAML, but it is still an attack-by-attack listing of countermeasures taken, not an overall analysis.) Further, they do not visibly follow robust-design principles [1, 2]. Indeed, the first version of Passport was vulnerable to a man-in-the-middle attack, and the current version is not as secure against that as it could be; vulnerabilities were also found in one of the original Liberty protocols (but not a browser-based one) and in permitted SAML instantiations [15, 6]. Earlier attacks on Passport were found in [9, 19]. They are mostly weaknesses of the environment, not of the protocol itself. Some of them apply to all browser-based protocols, and one has to keep in mind that this is a pretty vulnerable environment. In particular, online password guessing always works for remote wallets, fake wallet screens are a danger, and the protocol cannot get more secure than the underlying operating system and the browser. A brief security proof sketch for BBAE can be found in the preliminary version [13], but for a far less rigorous definition of BBAE. While one of our original motivations of the current detailed definition was to enable a more rigorous proof, we have not done this so far.

9   Names, Keys, and Certificates

We now recommend how to use identifiers (names) of users and names, keys, and certificates of wallets in the most important scenarios. Our abstract arbitrary authentication under an arbitrary name $name_W$ leaves that open, and so do concrete SAML responses, which have an arbitrary issuer $name_W$ and optional XML signatures. We concentrate on the general consumer scenario with many wallets and destination sites. Additionally, a wallet can use id_D to first look up whether it has a closer relationship with the destination site, e.g., a joint secret key, for more efficient authentication.

Form-filling case. The most usual case for initial e-commerce does not require authentication, because it replaces current forms that the user fills in without confirmation, e.g.,

with demographics and preferences. Even shipping addresses and payment information belong to this case, because the payment information is verified with existing payment systems. Then no wallet name and authentication are necessary. For a really anonymous response, a local wallet should choose a nonce as $name_W$ and make no signature.

Real identity. For authenticity under a name id_U^* with prior meaning, e.g., a legal name or an email address, this name must be verified in registration. Authenticity cannot get better than this verification, and all parties trusted by a destination site to confirm this name must be honest. (This is the same tradeoff between security and convenience as with PKIs; it has nothing to do with whether only a password is exchanged in registration or a public key.) Users with a local wallet have to get a public key pk of the wallet certified for id_U^*. The local wallet thus becomes a leaf in a tree of certification authorities for a specific, externally managed name space, where it may only confirm one or a few names.[6]

Long-term role. If a wallet generates a role name *role* for repeated use (e.g., with a certain group of enterprises), but without verified prior meaning or with the explicit desire to keep this role unlinkable to other roles, it generates a new name. (SAML explicitly allows this.) This is like SPKI/SDSI compared with classical PKIs. A remote wallet holder can issue responses for *role* under its own fixed name $name_W$ and with a fixed key pk without endangering unlinkability if its user community is large enough to offer anonymity. If the wallet is local and the role should really be unlinkable, it should generate a fresh name $name_{role}$ and key pk_{role} for this role. For XML signatures, the wallet can include this key in the KeyValue element when first using it with a destination site. The name *role* should be considered to be in a name space governed by pk or pk_{role}, respectively. Then only the wallet that issued *role* must be trusted for authenticity in this role.

To distinguish these cases precisely for the destination site, one should apply authentication context definitions similar to [10] to individual attributes in a response. For instance, one could represent the Passport case by stating that the user ID is a single long-term role, that names and payment information have not been verified, while control (but not ownership) of the email address has been verified.

10 Conclusion and Outlook

We have shown that browser-based attribute exchange and even the simpler single signon, although a type of three-party authentication, requires new protocols and poses new challenges for secure design. We presented a new protocol BBAE which, for the first time, combines all important privacy properties that can be achieved under the design constraints. Its efficiency is essentially the same as of other browser-based protocols.

A general trend in the protocol design for influential standards and products in this area is a multi-layered approach: One starts with a very general message format, then extends and refines it, then designs a small core protocol, and then extends that again.

---

[6] The wallet may use id_U also as its name *issuer*; this simplifies the use of X509 certificates. New XML tokens as certificates can be more flexible.

SAML already takes this approach to a certain extent. Liberty adds about one layer of extension between each layer of SAML. The WS-Federation proposals take this approach even further. The attractiveness for quick standardization is obvious. However, the challenges for finally getting secure protocols are large: First there is the simple practical issue that "general processing constraints" are spread over all layers, and both a security analyst and a later implementer may miss some of them or get them wrong. Secondly, the initial general message formats have no clear semantics, and even the core protocols are not always full protocols. Hence from a security point of view, the standards cannot currently be considered modular, because the first layer for which one can specify and prove usual security goals is typically the highest one. It will be a challenge to the security-research community to adapt to this trend.

Acknowledgements

References

1. M. Abadi, R. Needham: Prudent Engineering Practice for Cryptographic Protocols; IEEE Transactions on Software Engineering 22/1 (1996) 6–15
2. R. Anderson, R. Needham: Robustness Principles for Public Key Protocols; Crypto 95, Springer-Verlag, Berlin 1995, 236–247
3. K. Bohrer, X. Liu, D. Kesdogan, E. Schonberg, M. Singh, S. Spraragen: Personal Information Management and Distribution; 4th Intern. Conf. on Electronic Commerce Research (ICECR-4), Dallas, 2001
4. D. Chaum: Security without Identification: Transaction Systems to make Big Brother Obsolete; Communications of the ACM 28/10 (1985) 1030–1044
5. J. Camenisch, E. Van Herreweghen: Design and Implementation of the Idemix Anonymous Credential System; 9th ACM Conference on Computer and Communications Security (CCS), 2002, 21–30
6. Thomas Groß: Security Analysis of the SAML Single Sign-on Browser/Artifact Profile; 19th Annual Computer Security Applications Conference (ACSAC 2003), IEEE Computer Society Press, December 2003
7. IBM Consumer Wallet; White Paper, 1999 (first release 1997), http://www-3.ibm.com/software/webservers/commerce/payment/wallet.pdf
8. IBM: Enterprise Security Architecture using IBM Tivoli Security Solutions; April 2002, http://www.redbooks.ibm.com/abstracts/sg246014.html
9. D. P. Kormann, A. D. Rubin: Risks of the Passport Single Signon Protocol; Computer Networks 33 (2000) 51–58
10. Liberty Alliance Project: Liberty Phase 2 Final Specifications, November 2003, http://www.projectliberty.org/specs/lap-phase2-final.zip (v1.0 July 2002).
11. Microsoft Corporation: .NET Passport documentation, in particular Technical Overview, Sept. 2001, and SDK 2.1 Documentation (started 1999); http://www.passport.com and http://msdn.microsoft.com/downloads

12. Passlogix: v-Go Single Signon; White Paper, 2000 (first release 1999), http://www.passlogix.com/media/pdfs/usable_security.pdf
13. B. Pfitzmann, M. Waidner: BBAE – A General Protocol for Browser-based Attribute Exchange; IBM Research Report RZ 3455 (#93800) 09/09/02, http://www.zurich.ibm.com/security/publications/2002/
14. B. Pfitzmann, M. Waidner: Privacy in Browser-Based Attribute Exchange; ACM Workshop on Privacy in the Electronic Society (WPES) 2002, ACM Press 2003, 52–62
15. B. Pfitzmann, M. Waidner: Analysis of Liberty Single-Signon with Enabled Clients; IEEE Internet Computing 7(6) 2003, 38–44
16. Roboform: Free Web Form Filler and Password Manager; first release 1999, http://www.siber.com/roboform/.
17. Security Assertion Markup Language (SAML); OASIS Standard, Nov. 2002, http://www.oasis-open.org/committees/security/docs/
18. Shibboleth-Architecture Draft v05; May 2002 (v01 in 2001), http://middleware.internet2.edu/shibboleth/docs/draft-internet2-shibboleth-arch-v05.pdf
19. M. Slemko: Microsoft Passport to Trouble; Rev. 1.18, Nov. 2001 http://alive.znep.com/ marcs/passport/
20. BEA, IBM, Microsoft, RSA Security, VeriSign: WS-Federation: Passive Requestor Profile; Draft, Version 1.0, July 2003, http://www-106.ibm.com/developerworks/webservices/
21. HTTP Over TLS; Internet RFC 2818, 2000
22. XML-Signature Syntax and Processing; W3C Recommendation, Feb. 2002, http://www.w3.org/TR/xmldsig-core/
23. Zeroknowledge: Freedom Personal Firewall; first release 1999, http://www.freedom.net/products/firewall/index.html