CardSpace-Liberty Integration for CardSpace Users

Haitham S. Al-Sinani[*]
Information Security Group
Royal Holloway, University of London
http://www.isg.rhul.ac.uk
H.Al-Sinani@rhul.ac.uk

Waleed A. Alrodhan
Information Security Group
Royal Holloway, University of London
http://www.isg.rhul.ac.uk
W.A.Alrodhan@rhul.ac.uk

Chris J. Mitchell
Information Security Group
Royal Holloway, University of London
http://www.isg.rhul.ac.uk
C.Mitchell@rhul.ac.uk

ABSTRACT

Whilst the growing number of identity management systems have the potential to reduce the threat of identity attacks, major deployment problems remain because of the lack of interoperability between such systems. In this paper we propose a novel scheme to provide interoperability between two of the most widely discussed identity management systems, namely Microsoft CardSpace and Liberty. In this scheme, CardSpace users are able to obtain an assertion token from a Liberty-enabled identity provider that will satisfy the security requirements of a CardSpace-enabled relying party. We specify the operation of the integration scheme and also describe an implementation of a proof-of-concept prototype. Additionally, security and operational analyses are provided.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and protection

General Terms

Security

Keywords

Identity Management, CardSpace, Liberty Alliance Project, Interoperability, SAML, Browser Extension

1. INTRODUCTION

In line with the continuing increase in the number of online services requiring authentication, there has been a proportional rise in the number of digital identities needed for authentication purposes. This has contributed to the recent rapid growth in identity-oriented attacks, such as phishing, pharming, etc. In an attempt to mitigate such attacks, a number of identity management systems have been proposed.

Identity management deals with uniquely identifying individuals in a system, and with effectively controlling access to the system resources by managing the rights and privileges associated with digital identities. The most important service provided by an identity management system is authentication. Such a system may also support other services, such as pre-authentication, authorisation, single sign-on, identity repository management, user self-service registration, and audit. Examples of identity management systems include CardSpace[1], Liberty[2], OpenID[3], and Shibboleth[4] [5, 8, 17, 46, 50].

Most identity management architectures involve the following main roles.

1. The identity provider (IdP), which issues an identity token to a user.

2. The service provider (SP), or the relying party (RP) in CardSpace terminology, which consumes the identity token issued by the IdP in order to identify the user, before granting him/her access.

3. The user, also known as the principal.

4. The user agent, i.e. software employed by a user to send requests to webservers and receive data from them, such as a web browser. Typically, the user agent processes protocol messages on behalf of the user, and prompts the user to make decisions, provide secrets, etc.

An identity provider supplies a user agent with an authentication token that can be consumed by a particular service provider. Whilst one service provider might solely support CardSpace, another might only support Liberty. Therefore, to make these systems available to the largest possible group of users, effective interoperability between systems is needed. In this paper we investigate a case involving a CardSpace-enabled relying party, a Liberty-enabled identity provider, and a user agent that is (only) CardSpace-enabled. The goal is to develop an approach to integration that is as transparent as possible to both identity providers and relying parties.

[*]This author is sponsored by the Diwan of Royal Court, Sultanate of Oman.

[1]`http://msdn.microsoft.com/en-us/library/aa480189.aspx`
[2]`http://www.projectliberty.org/`
[3]`http://openid.net/`
[4]`http://shibboleth.internet2.edu/`

We have chosen to consider the integration of Liberty with CardSpace because of Liberty's wide adoption (see section 2.2.1). Currently, it is a leading identity management architecture, that has gained the acceptance of a number of technology-leading companies and organisations. Complementing this, the wide use of Windows, recent versions of which incorporate CardSpace, means that enabling interoperation between the two systems is likely to be of significance for large numbers of identity management users and service providers. Another reason for choosing Liberty is because of the similarity between the message flows in its ID-FF profile and CardSpace.

The remainder of the paper is organised as follows. Section 2 presents an overview of CardSpace and Liberty, and section 3 contains the proposed integration scheme. In section 4, we provide an operational analysis of the scheme and, in section 5, we describe a prototype implementation. Section 6 highlights possible areas for related work, and, finally, section 7 concludes the paper.

2. CARDSPACE AND LIBERTY

We provide an introduction to the CardSpace and Liberty identity management systems. SAML is also briefly outlined.

2.1 CardSpace

We first give a general introduction to CardSpace, covering relevant operational aspects.

2.1.1 Introduction to CardSpace

CardSpace is Microsoft's implementation of a digital identity metasystem, in which users can manage digital identities issued by a variety of identity providers, and use them in a range of contexts to access online services. In CardSpace, digital identities are represented to users as Information Cards (or InfoCards). From the CardSpace perspective, InfoCards are XML-based files that list the types of claim made by one party about itself or another party. CardSpace is designed to reduce reliance on username-password authentication, and to provide a consistent authentication experience across the Web to improve user understanding of the authentication process. It is claimed that CardSpace is also designed to reflect the seven identity laws promulgated by Microsoft [6, 10, 17, 34].

The concept of an InfoCard is inspired by real-world cards, such as driving licences and credit cards. A user can employ one InfoCard with multiple websites. Alternatively, just as different physical ID cards are used in distinct situations, separate InfoCards can be used at different websites, helping to enhance user privacy and security. If InfoCards are obtained from different IdPs, the credentials referred to by such cards are stored in distinct locations, potentially improving reliability and security, as well as giving users flexibility in choosing points of trust.

There are two types of InfoCards: personal (self-issued) cards and managed cards. Personal cards are created by users themselves, and the claims listed in such an InfoCard are asserted by the self-issued identity provider (SIP) that co-exists with the CardSpace identity selector on the user machine. In this paper we use personal cards to enable interoperation between CardSpace and Liberty. Managed cards, on the other hand, are obtained from remote identity providers.

The InfoCards themselves do not contain any sensitive information; instead an InfoCard carries metadata that indicates the types of personal data that are associated with this identity, and from where assertions regarding this data can be obtained. The data referred to by personal cards is stored on the user machine, whereas the data referred to by a managed card is held by the identity provider that issued it [6, 16, 18, 24, 34, 35, 38].

By default, CardSpace is supported in Internet Explorer (IE) from version 7 onwards. Extensions to other browsers, such as Firefox[5], and Safari[6] also exist. Microsoft has recently released an updated version of CardSpace, known as Windows CardSpace 2.0 Beta 2[7]. However, in this paper we refer throughout to the CardSpace version that is shipped by default as part of Windows Vista and Windows 7, which has also been approved as an OASIS standard under the name 'Identity Metasystem Interoperability Version 1.0' (IMI 1.0) [28].

2.1.2 CardSpace Personal Cards

The core idea introduced in this paper is to use CardSpace personal cards to make Liberty identity providers available via the CardSpace identity selector. We therefore next describe CardSpace personal cards.

Creation of Personal Cards.

Prerequisites for use of a CardSpace personal card include:

1. a CardSpace-enabled RP; and
2. a CardSpace-enabled user agent, e.g. a web browser capable of invoking the CardSpace identity selector, such as those shipped as part of Windows Vista and Windows 7.

The identity selector allows a user to create a personal card and populate its fields with self-asserted claims. To protect users from disclosing sensitive information, CardSpace restricts the contents of personal cards to non-sensitive data, such as that published in telephone directories. Personal cards currently only support 14 editable claim types, namely *First Name, Last Name, Email Address, Street, City, State, Postal Code, Country/Region, Home Phone, Other Phone, Mobile Phone, Date of Birth, Gender*, and *Web Page*. Data inserted in personal cards is stored in encrypted form on the user machine.

When a user creates a new personal card, CardSpace generates an ID and a master key for this card. The card ID is a globally unique identifier (GUID), and the master key is 32 bytes of random data.

Using Personal Cards.

When using personal cards, CardSpace adopts the following protocol. We describe the protocol for the case where the RP does not employ a security token service (STS[8]).

1. User agent → RP. HTTP/S request: GET (login page).

[5]`https://addons.mozilla.org/en-US/firefox/addon/10292`

[6]`http://www.hccp.org/safari-plug-in.html`

[7]`http://technet.microsoft.com/en-us/library/dd996657(WS.10).aspx`

[8]The STS is responsible for security policy and token management within an IdP and, optionally, within an RP [27].

2. RP → user agent. HTTP/S response. A login page is returned containing the CardSpace-enabling tags in which the RP security policy is embedded.

3. User → user agent. The user agent offers the user the option to use CardSpace (e.g. via a button on the RP web page); selection of this option causes the agent to invoke the CardSpace identity selector, passing the RP policy to the selector. Note that if this is the first time that this RP has been contacted, the identity selector will display the identity of the RP, giving the user the option either to proceed or to abort the protocol.

4. User agent → user agent (identity selector → InfoCards). The CardSpace identity selector, after evaluating the RP security policy, highlights the InfoCards that match the policy, and greys out those that do not. InfoCards previously used for this particular RP are displayed in the upper half of the selector screen.

5. User → user agent (user → identity selector). The user chooses a personal card. (Alternatively, the user could create and choose a new personal card). The user can also preview the card (with its associated claims) to see which claim values are being released. Note that the selected InfoCard may contain several claims, but only the claims explicitly requested in the RP security policy will be passed to the requesting RP.

6. User agent $\rightleftharpoons$ user agent (identity selector $\rightleftharpoons$ SIP). The identity selector creates and sends a SAML-based Request Security Token (RST) to the SIP, which responds with a SAML-based Request Security Token Response (RSTR).

7. User agent → user agent (identity selector → user agent). The RSTR is then passed to the user agent, which forwards it to the RP.

8. RP → user. The RP validates the token, and, if satisfied, grants access to the user.

The managed card operational protocol is similar, except that the remote IdP specified in the InfoCard is contacted instead of the SIP. The CardSpace identity selector then uses the standard identity metasystem protocols (see section 2.1.3) to first retrieve the IdP security policy[9] and then obtain a security token representing the selected digital identity from the STS of the remote IdP. The identity selector then passes the received token to the user agent, optionally after first obtaining permission from the user[10] [27, 41].

For CardSpace to work, both the RP and the IdP must be CardSpace-enabled. The problem that we address here is the incompatibility issue that will occur if the RP is CardSpace-enabled whereas the IdP is not, but is instead Liberty-enabled. Addressing this issue could help to extend the applicability of CardSpace.

[9]Depending on the IdP security policy, the user may be requested to provide credentials for authentication to the selected IdP. The authentication methods currently supported by CardSpace include username-password authentication, a KerberosV5 service ticket, an X.509v3 certificate, and a self-issued token.

[10]This may involve presenting the user with a 'display token', prepared by the remote IdP, listing the claim values asserted in the 'real' security token; the identity selector will only continue if the user is willing to release such values.

Private Personal Identifiers.

The private personal identifier (PPID) is a unique identifier linking a specific InfoCard to a particular RP [6, 7, 38]. CardSpace RPs can use the PPID along with a digital signature to authenticate a user.

When a user uses a personal card at an RP for the first time, CardSpace generates a site-specific:

- PPID by combining the card ID with data taken from the RP certificate; and
- signature key pair by combining the card master key with data taken from the RP certificate.

In both cases, the domain name or IP address of the RP is used if no RP certificate is available.

Since the PPID and key pair are RP-specific, the PPID does not function as a global user identifier, helping to enhance user privacy. In addition, compromising the PPID and key pair for one RP does not allow an adversary to impersonate the user at other RPs. The CardSpace identity selector only displays a shortened version of the PPID to protect against social engineering attacks and to improve readability.

When a user first registers with an RP, the RP retrieves the PPID and the public key from the received authentication token, and stores them. If a personal InfoCard is re-used at a site, the supplied authentication token will contain the same PPID and public key as used previously, signed using the corresponding private key. The RP compares the received PPID and public key with its stored values, and verifies the digital signature. If all checks succeed it has assurance that it is the same user.

The PPID could be used on its own as a shared secret to authenticate a user to an RP. However, it is recommended that the associated (public) signature verification key, as held by the RP, should also always be used to verify the signed authentication token to provide a more robust authentication method [6].

2.1.3 CardSpace Protocols

In order to maximise interoperability with non-Windows platforms, CardSpace has been specifically designed to use open standards-based protocols, notably the WS-* standards, the most significant of which are listed below.

WS-Policy/WS-SecurityPolicy is used to describe security policies [3, 21]. Note that a website can also describe its policy in HTML/XHTML.

WS-MetadataExchange is used to fetch security policies and exchange service description metadata over the Internet [4]. Note that a website can also transmit its security policy using HTTP/S.

WS-Trust is used to acquire security tokens (e.g. SAML tokens) from IdPs [2].

WS-Security is used to securely deliver security tokens to RPs [37]. Note that HTTP/S can also be used.

2.1.4 Proof Keys

A SAML security token can be coupled with cryptographic evidence to demonstrate the sender's rightful possession of the token. A 'proof key' is a key associated with a security token, and the data string used to demonstrate the sender's knowledge of that key (e.g. through the inclusion of a digital signature or MAC computed using the key) is called the 'proof-of-possession' of the security token [27, 38].

A security token can be associated with two types of proof key.

1. Symmetric proof keys

   If a symmetric key token is requested, a symmetric proof key is established between the identity selector and the CardSpace-enabled IdP [38], which is then revealed to the RP. This key is used to prove the subject's rightful possession of the security token. Whilst the use of such a key may optimise token processing in terms of speed and efficiency [36], it involves revealing the identity of the RP to the IdP, which is not ideal from a privacy perspective.

2. Asymmetric proof keys

   If an asymmetric key token is requested, the identity selector generates an ephemeral RSA key pair and sends the public part of the key to the CardSpace-enabled IdP. The identity selector also sends a supporting signature to prove ownership of the corresponding private key [38]. If approved by the IdP, the public part is sent to the RP in the security token. The private part of the RSA key pair is then used to prove the subject's rightful possession of the security token. Although the use of such a key may not be as efficient as the symmetric approach, it helps to protect user privacy since the identity of the RP does not need to be disclosed to the IdP.

It merits mentioning that the default behaviour of the CardSpace identity selector is different in the special case of browser-based client interactions with a website, in which case 'bearer' tokens are requested. Because a web browser is only capable of submitting a token to a website passively over HTTP without any proof-of-possession, bearer tokens with no proof keys are used [36].

2.2 Liberty

We next give a general introduction to Liberty, covering relevant operational aspects.

2.2.1 *Introduction to Liberty*

The Liberty Alliance is a large consortium, established in 2001 by approximately 30 organisations; it now has a global membership of more than 150[11]. The Liberty Alliance Project (or simply Liberty) builds open, standards-based specifications for federated identity, provides interoperability testing, and helps to prevent identity theft. Liberty also aims to establish best practices and business guidelines for identity federation. According to its website, Liberty has been widely adopted with, as of 2006, more than one billion Liberty-enabled identities and devices[12]. As of mid 2009, the work of the Liberty Alliance is being adopted by the Kantara Initiative[13].

Figure 1 shows the general Liberty model, which is essentially a single sign-on (SSO) model [11]. In this model, a principal (or a user) can federate its various identities to a single identity issued by an identity provider, so that the user can access services provided by service providers belonging to the same circle of trust by authenticating just once to the identity provider. This relies on a pre-established relationship between the identity provider and every service provider in the circle of trust.

[11]`http://www.projectliberty.org/liberty/membership/current_members/`

[12]`http://www.projectliberty.org/liberty/adoption/`

[13]`http://kantarainitiative.org/`

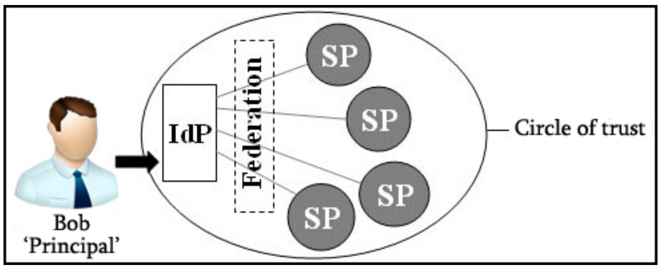


Figure 1: The Liberty model

The Liberty specifications are divided into three frameworks: the identity federation framework (ID-FF) [49], the identity web services framework (ID-WSF) [47] and the service interface specifications (ID-SIS) [30]. In this paper we focus on the ID-FF. The ID-FF provides approaches for implementing federation and SSO, including supporting mechanisms such as session management and identity/account linkage.

2.2.2 *Liberty Functional Requirements*

The Liberty architecture [49] supports the following activities.

Identity federation This is the process of linking a user's SP identity with a specific IdP (given user consent). At the time of federation, two user pseudonyms[14] are created for the IdP-SP association, one for use by each party. De-federation is the reverse process.

Single sign-on This feature enables a user to log in once to an IdP in a Liberty circle of trust and subsequently use SPs belonging to this circle without the need to log in again. Global log-out is the reverse process.

Anonymity A Liberty SP may request a Liberty IdP to supply a temporary pseudonym that will preserve the anonymity of a user. This identifier may be used to obtain information for or about the user (given their consent) without requiring the user to consent to a long term relationship with the SP [49].

2.2.3 *Single Sign-on and Federation Profiles*

The Liberty ID-FF protocol specification [14] defines the SSO and federation protocol. The ID-FF bindings and profile specification [12] defines profiles, i.e. mappings of ID-FF protocol messages to particular communication protocols (e.g. HTTP [22]). The latter document also describes the common interactions and processing rules for these profiles.

The single sign-on and federation protocol has three associated profiles, summarised below.

[14]A pseudonym is an opaque but unique handle (identifier) for the user, enabling the user's real identity to remain private. Pseudonyms can be temporary or persistent, and are included in SAML tokens exchanged between a Liberty IdP and SP.

Liberty artifact profile The Liberty artifact profile involves embedding an artifact (i.e. an opaque handle) in a URI exchanged between the IdP and SP via Web redirection, and also requires direct (background) communication between the SP and IdP [49]. The SP uses the artifact to retrieve the full SAML assertion from the IdP. As it requires direct SP-IdP communication, which is inconsistent with the CardSpace approach[15], the proposed scheme does not support this profile.

Liberty browser post profile JavaScript-enabled browsers can perform an HTTP redirect between IdPs and SPs by using JavaScript to automatically send a form (containing the authentication data). This profile embeds the entire SAML assertion in an HTML form. As a result, it does not use an artifact and does not require any direct communication between the SP and the IdP. The scheme proposed here supports this profile.

Liberty-enabled client (and proxy) profile This profile defines interactions between Liberty-enabled clients (and/or proxies), SPs, and IdPs. A Liberty-enabled client (LEC) is a user agent that can directly communicate with the IdP that the user intends to use to support its interactions with an SP. In addition, the LEC sends and receives Liberty messages in the body of HTTP requests/responses using 'post', rather than relying upon HTTP redirects and encoding protocol parameters into URLs. Therefore, LECs do not impose any restrictions on the size of the protocol messages. Interactions between a user agent and an IdP are SOAP-based, and the protocol messages include Liberty-specified HTTP headers.

Although it adds complexity, this profile seems like a natural fit to the proposed scheme. We propose to use the CardSpace identity selector to act as a Liberty-enabled client. In our scheme, the identities of the IdPs are stored on CardSpace personal cards.

2.2.4 *Proof Keys*

The Liberty ID-FF supports SAML 2.0 assertions as a security token type. The SAML 2.0 specifications offer three proof-of-possession methods (also referred to as subject confirmation methods): Holder-of-Key (HoK), Sender-Vouches, and bearer [13].

The HoK method [45] can be used to address both the symmetric and asymmetric proof-of-possession requirements of a CardSpace-enabled RP.

2.3 SAML

SAML is an XML-based standard for exchanging identity-related information across the Internet. The SAML specifications cover four major elements.

A SAML assertion can contain three types of statement:

1. an authentication statement, asserting that a user was authenticated at a particular time using a particular authentication method;
2. an attribute statement, asserting that a user is associated with certain attributes; and
3. an authorization decision statement, asserting that a particular user is permitted to perform a certain action on a specific resource.

SAML protocols define data structures for sending SAML requests and returning assertions.

SAML bindings map SAML protocol messages onto standard communication protocols, e.g. HTTP.

SAML profiles describe how SAML assertions, protocols and bindings are combined together to support a particular use case.

SAML 1.0 [26] was first adopted as an OASIS standard in 2002; a minor revision, SAML 1.1 [33], was formally adopted in 2003. A major revision led to SAML 2.0 [13], which became a standard in 2005. The differences[16] between version 1.1 and 2.0 are significant, and SAML assertions of the two types are incompatible.

Finally note that the CardSpace SIP currently only issues tokens conforming to SAML 1.1 [38], whereas the Liberty specifications require IdPs to generate assertions using SAML 2.0 syntax.

3. THE INTEGRATION SCHEME

This section provides an overview of the scheme, and also gives a brief description of its protocol flow. However, we first highlight the main differences between the integration scheme proposed here and a previously proposed scheme of this type.

3.1 Previous Work

The integration scheme proposed here builds on a previous proposal for CardSpace-Liberty integration [1], referred to below as the AM scheme. Whilst the scheme proposed here has some properties in common with this previous proposal, for example both approaches concentrate on supporting integration at the client rather than at the server, there are a number of important differences.

Instead of focusing on CardSpace users only, as is the case with the scheme described here, the AM scheme allows for full interoperability even in the case where the SP is Liberty-enabled and the IdP is CardSpace-enabled. However, since no prototype has been developed, issues which might arise during deployment have not been explored. By contrast, the scheme described below has been prototyped, and hence greater confidence can be derived in its practicality.

One important goal for any identity management system is ease of use. However, user interface issues, notably the operation of the integration software on the client platform, have not been explored for the AM scheme, whereas the proposal here addresses this through a combination of a browser extension and the CardSpace interface. In addition, whereas the relationship between the integration software and the web browser is not specified for the AM scheme, this issue has been resolved for the scheme presented here by implementing the functionality in a web browser plug-in residing on the user machine.

The means by which the integration software is triggered is also not clear for the AM scheme. For example, if the integration software is assumed to run at all times, then

[15]In CardSpace, all RP-IdP communications must go through the identity selector on the user machine.

[16]`https://spaces.internet2.edu/display/SHIB/SAMLDiffs`

problems arise if the user wants to use CardSpace or Liberty without integration. By contrast, several ways of addressing this particular issue are described in sections 3.2.3 and 4.3.

The AM scheme does not address how to handle the private personal identifier (PPID), described in section 2.1.2, when supporting interoperation between RPs and Liberty-enabled IdPs. Additionally, it is not clear whether providing the full address of the IdP is the responsibility of the RP, the integration software, or the user. These issues are addressed in sections 3.2 and 5.

3.2 Integration Protocol

We now present the novel protocol.

3.2.1 System Parties

As stated earlier, the integration scheme addresses the incompatibility issue arising if the RP is CardSpace-enabled and the IdP is Liberty-enabled. The parties involved are as follows.

1. A CardSpace-enabled RP.
2. A CardSpace-enabled user agent (e.g. a suitable web browser).
3. A Liberty-enabled IdP.
4. The integration browser extension (which must first be installed).

Note that there is no need for a Liberty-enabled user agent. Instead the user only needs to install the integration browser extension.

Figure 2 gives a simplified picture of the high-level interactions between system parties on the user machine. The parties shown are the browser extension, the user agent (browser), the identity selector, and the SIP. The arrows indicate information flows.

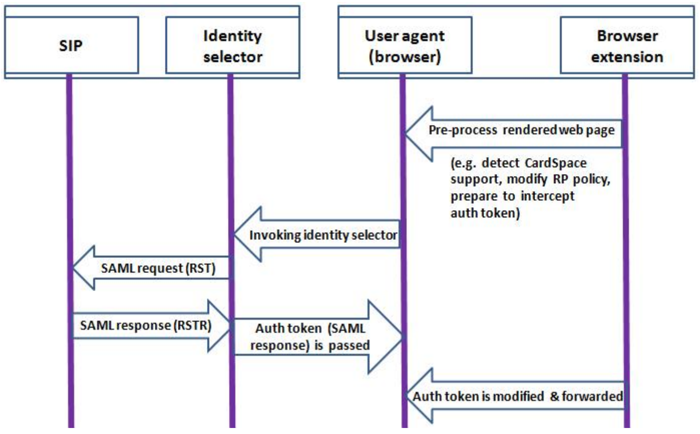

SIP
Identity selector
User agent (browser)
Browser extension
Pre-process rendered web page
(e.g. detect CardSpace support, modify RP policy, prepare to intercept auth token)
Invoking identity selector
SAML request (RST)
SAML response (RSTR)
Auth token (SAML response) is passed
Auth token is modified & forwarded

Figure 2: Data flows between client parties

3.2.2 Preconditions

The scheme has the following requirements.

- The user must have an existing relationship with a CardSpace RP.
- The user must have an existing relationship with a Liberty-enabled IdP, and hence the IdP has a means of authenticating the user.
- The CardSpace-enabled RP must not employ an STS (see section 4.7). Instead, the RP must express its security policy using HTML/XHTML, and interactions between the CardSpace identity selector and the RP must be based on HTTP/S via a web browser. This is because of the use of a browser extension (see section 3.2.4) in the scheme, and a browser extension by itself is incapable of managing the necessary communications with an STS.
- The CardSpace-enabled RP must support SAML 2.0 (see section 2.3).
- As well as being able to verify the InfoCard signature, the CardSpace-enabled RP must be able to verify the the IdP digital signature in the provided SAML token.
- The Liberty-enabled IdP must be prepared to provide SAML assertions for SPs for which a federation agreement does not exist for the user concerned[17]. In the absence of the IdP-SP-specific user pseudonyms (which would exist if federation had occurred) the IdP is prepared to use the InfoCard PPID for the user in place of Liberty pseudonyms in the SAML request and response messages (and in the created SAML assertion). This avoids changes to the Liberty message formats, but does require a minor policy/operational change to the Liberty-enabled IdP.

3.2.3 LibertyCards

Either prior to, or during, use of the integration protocol, the user must create a special personal card, referred to as a LibertyCard, which will represent the Liberty IdP. This card must contain the URL of the Liberty IdP it represents, and must also contain a predefined sequence of characters, e.g. the word 'Liberty', which will be used to trigger the integration software (see section 4.3).

The browser extension, described in section 3.2.4, must process the policy statement provided by the RP before it is passed to the identity selector. It must first decide whether or not the RP policy requirements can be met by one or more of the LibertyCards; if not then it leaves the policy statement unchanged, and the browser extension plays no further active part in processing. However, if use of a LibertyCard is appropriate, then the browser extension changes the policy to include the types of claim employed by LibertyCards. For example, if the URL of the Liberty IdP is stored in the *web page* field of the LibertyCard, then the browser extension must modify the RP security policy to add the *web page* claim (see section 5.3.1 for further details). Note that adding the claim types to the RP security policy is necessary to ensure that the token supplied by the SIP contains the values of these claims, which can then be processed by the browser extension; otherwise these values would not be available to the browser extension[18].

[17]It is thus not necessary for the user to Liberty-federate the IdP with the RP (which would in any case be difficult to achieve given that we are not requiring the RP to be Liberty-enabled).

[18]Unfortunately, whilst necessary for the operation of the browser extension, adding claims to the RP policy means that CardSpace-compliant IdPs for which the user has 'managed' InfoCards, and which might otherwise be acceptable to the RP, cannot be selected by the user.

One approach that would avoid the need to store the URL of the IdP in a personal card would involve the browser extension prompting the user to enter the URL of the IdP that they wish to contact, after they have selected a card. This could occur as part of step 8 in section 3.2.5. However this approach is not adopted here because it would require the user to manually enter the URL every time a LibertyCard is used, causing usability issues.

3.2.4 Browser Extension

The integration scheme is based on a browser extension that is able to:

- automatically execute;
- read and inspect browser-rendered web pages;
- modify rendered web pages if certain conditions hold;
- intercept, inspect and modify messages exchanged between a CardSpace identity selector and a CardSpace-enabled RP (via a browser);
- automatically forward security tokens (via browser-based HTTP redirects) to Liberty-enabled IdPs and to CardSpace-enabled RPs; and
- provide a means for a user to enable or disable it.

3.2.5 Protocol Operation

Figure 3 gives a simplified sketch of the integration scheme. The protocol operates as follows (with step numbers as shown in figure 3). Steps 1, 2, 4–7 and 12 of the integration scheme are the same as steps 1, 2, 3–6 and 8, respectively, of the CardSpace personal card protocol given in section 2.1.2, and hence are not described again here.

3. User agent → user agent (browser extension → browser). The browser extension scans the login page to detect whether the RP website supports CardSpace. If so, it starts to process the browser-rendered login page, including embedding a function into the page to intercept the authentication token that will later be returned by the CardSpace identity selector. If not, the browser extension terminates.

8. User agent → user agent (identity selector → browser extension). Unlike in the 'standard' case, the RSTR is not sent to the RP; instead the browser extension intercepts the RSTR (a SAML authentication response), converts it into a SAML authentication request, and forwards it to the appropriate Liberty-enabled IdP. Note that the detailed format of the SAML authentication request will depend on the Liberty profile being used (see discussion below).

9. Liberty-enabled IdP $\rightleftharpoons$ user. If necessary, the Liberty-enabled IdP authenticates the user.

10. Liberty-enabled IdP → user agent. The IdP sends a SAML authentication response to the user agent. This response is also Liberty profile-dependent (see discussion below).

11. User agent → RP. The user agent forwards the token to the RP, optionally after first obtaining permission from the user (see section 4.4).

The detailed operation of steps 8 and 10 is dependent on the Liberty profile in use between the user agent and the IdP. The construction of the SAML authentication request in step 8 differs depending on whether the Liberty browser post (LBP) profile or the Liberty-enabled client (LEC) profile is in use. For example, the URI identifier 'URI: `http://projectliberty.org/profiles/brws-post`' must be used when employing the LBP profile, whereas 'URI: `http://projectliberty.org/profiles/lecp`' must be used when employing the LEC profile. In addition, when using the LEC profile, the authentication request must be submitted to the IdP as a SOAP [25] request with a Liberty-enabled header, whereas when using LBP, the authentication request to the IdP can be embedded in an HTML form.

The details of steps 10 and 11 differ significantly depending on which of the two Liberty profiles is in use. In the LEC profile, in step 10 the IdP returns the authentication response to the client (which is responsible for forwarding it to the specified SP). In the LBP profile, however, the IdP sends the HTML form carrying the authentication response to the user agent, and redirects the user via the user agent to the specified SP. Such a procedure would deny the browser extension the opportunity to intercept the communication and give the user the choice whether or not to allow the token to be sent to the RP (as is normally the case for CardSpace). We therefore require a small modification to the way that the Liberty-enabled IdP operates. The IdP must be modified to redirect the user agent to a web page at the IdP server, rather than at the RP, thereby giving the browser extension control. This could be achieved by requiring the IdP to set the action attribute[19] of the HTML form to an empty string or to #[20]. In step 11, the browser extension resets the action attribute to the URL address of the appropriate CardSpace RP, and, after obtaining user permission to release the authentication token to the given RP, automatically submits the HTML form, redirecting the user agent to the RP website. This small change to the normal operation of the Liberty IdP helps to enhance user control (see sections 4.4 and 5.3.3), hence implementing Microsoft's first identity law [6, 10, 17, 34]. It merits mentioning that both the LBP and LEC profiles require the SP URL address to be specified as the value of the '<lib:AssertionConsumerServiceURL>' statement in the SAML authentication request [12]. To keep the changes at the IdP side to a minimum, the value of this field could be set to #, implicitly instructing the IdP to include this value instead of the SP's URL in the action attribute of the HTML form sent back to the user agent. Further discussion of the LBP and LEC profiles is given in section 4.2.

Given that we have assumed that the RP supports SAML 2.0 tokens, there is no need to modify the proof-of-possession data since the RP can use the Liberty ID-FF supported HoK [45] method (which can be symmetric or asymmetric) to express its proof-of-possession requirements. However, a

[19]Observe that, in the standard LBP profile case, the action attribute of the HTML form is set to the URL address of the requesting SP, and the IdP redirects the user agent to that SP.

[20]Note that whilst this has been shown to work successfully with IE7 and IE8, other browsers may not support an action attribute of an empty string or hash (#); hence setting the action attribute to a relative URL for the IdP login page may be required for such browsers.

symmetric proof key should only be used if the user is willing to disclose the identity of the RP to the IdP, and if the RP holds a valid certificate. For browser-based applications (and also where no proof-of-possession is needed), the proposed scheme supports bearer tokens [13, 36, 38].

Finally observe that the additional steps above can be integrated into the current CardSpace framework relatively easily, as the prototype implementation shows.

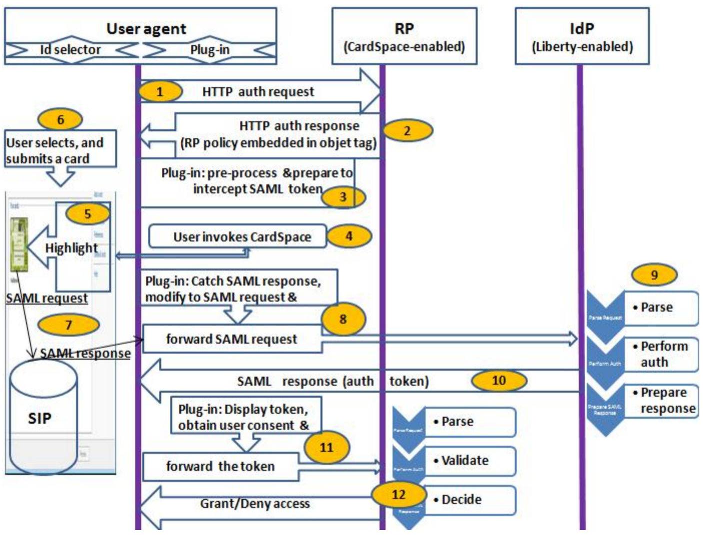


Figure 3: Protocol exchanges

4. DISCUSSION AND ANALYSIS

We now consider implementation and applicability issues of the scheme.

4.1 Differences in Scope

There is a key difference between the Liberty ID-FF and CardSpace frameworks. CardSpace allows IdPs to assert a range of attributes about users (including simple authentication assertions), whereas Liberty ID-FF only supports authentication assertions. In CardSpace, the user attributes to be asserted are specified in a SAML attribute statement contained in a SAML request that can be processed by the local SIP or the remote CardSpace-enabled IdP. However, a Liberty ID-FF conformant IdP is only required to generate SAML authentication statements (and not assert user attributes), which gives rise to an interoperation problem. Two possible solutions are as follows.

1. It could be assumed that the CardSpace RP is only concerned with user authentication (which seems likely to be a common case). In such a case a LibertyCard contains the IdP URL and the trigger word, and a LibertyCard will only be used if the RP policy requests an assertion solely of the PPID attribute, e.g. by including '`http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier`' in the list of required claims. In such a case, the browser extension will modify the RP policy to ensure it includes the fields used in LibertyCards (see section 3.2.3). On selection of a LibertyCard, the browser extension (as in step 8 in section 3.2.5) intercepts, creates and forwards a SAML authentication request to the user-selected IdP. While this is a straightforward task, it limits the scope of applicability of the scheme.

2. Alternatively, it could be assumed that the CardSpace-enabled RP is concerned with both user authentication and the assertion of user attributes, and that the RP policy permits assertions (for user attributes only) to be provided by the SIP. In this case, along with requiring the PPID, the RP security policy would also specify the attributes required, leading the identity selector to highlight the user-created LibertyCards that satisfy the requirements. To ensure that no changes are required at either the RP or the IdP, the browser extension could store attribute assertions created by the SIP. The browser extension would then create the SAML authentication request according to the Liberty ID-FF standards, and forward it to the specified IdP. When the browser extension receives the response containing the authentication assertion from the IdP, it would add appropriate attribute assertion(s) from its local cache and then forward the entire package to the RP. However, if the RP security policy dictates that security tokens must be wholly signed by the issuing IdP, then this solution would fail.

The prototype implementation, described in section 5, implements the first approach.

4.2 Liberty Profiles

To maximise applicability, the integration scheme supports both the Liberty browser post (LBP) and Liberty-enabled client (LEC) profiles, introduced in section 2.2.3. However, the prototype described in section 5 only implements the LBP profile.

In the LEC profile, interactions between a user agent and an IdP are SOAP-based, and the protocol messages include Liberty-specified HTTP headers indicating that the sender is Liberty-enabled. Under the LEC profile, the client must submit the authentication request to the IdP as a SOAP request, whereas, when using the LBP profile, the request can be embedded in an HTML form containing a field called 'LAREQ' set to the '<lib:AuthnRequest>' protocol message [12, 14]. In order to support both profiles, the integration software must therefore be capable of supporting both forms of communications with the IdP.

The two profiles have many properties in common. For example, they both support SAML. In both profiles, the HTML form containing the authentication response must be sent to the user agent using an HTTP POST; this form must contain the field 'LARES' with value equal to the authentication response, as defined in the Liberty protocol schema [14]. In both profiles, the value of the 'LARES' field must be encoded using a base-64 transformation [23].

Despite the differences between the profiles, the protocol steps given in section 3.2.5 apply to both profiles.

4.3 Triggering the Browser Extension

As stated in section 3.1, the means by which the integration software is triggered needs to be chosen carefully. The means included in the scheme described in section 3.2.3 is to include a trigger sequence (e.g. the word 'Liberty') in a specific field of a LibertyCard. This is also the method used in the prototype described in section 5. However, other approaches could be used, e.g. as follows.

1. The browser extension could start whenever CardSpace is triggered. When a user submits an InfoCard, the browser extension would offer the user two options (based on HTML forms): to continue to use CardSpace as usual, or to use a Liberty-enabled IdP. This approach gives a greater degree of user control, and hence implements Microsoft's first identity law [6, 10, 17, 34]. However, it is not particularly convenient, since it would always require users to choose whether or not to use the integration software.

2. Alternatively, the browser extension could ask the user whether they wish to activate the integration protocol (e.g. via a JavaScript pop-up box). This has advantages and disadvantages similar to those of the first alternative.

4.4 Token Forwarding

The means by which the security token is forwarded to the RP needs to be chosen carefully. We refer to the numbered protocol steps given in section 3.2.5.

The responsibility for delivering the security token could be given to the Liberty IdP (as is normally the case when using the LBP profile). In this case the RP address could be added to the SAML authentication request (as prepared in step 8) so that the IdP knows which RP it must forward the token to (again as is normally the case for the Liberty profiles). Although this would avoid the need for changes to the normal operation of the Liberty IdP and potentially also help auditing, such an approach has privacy implications since the IdP would learn the identity of the RP.

As a result, as specified in step 11 of the proposed scheme, the responsibility for sending the security token to the RP is given to the user agent. Thus a means is required for giving the browser extension the address of the RP, so that it can forward the token. We next consider three possible ways in which the RP address might be made available.

- The RP address could be stored in the browser extension itself. Whilst this puts the user in control, it is not user-friendly, as it would require users to manually add the address of each RP into the code of the browser extension.

- After the security token is returned from the Liberty IdP, the browser extension could ask the user to enter the RP address, e.g. using a JavaScript pop-up box or an HTML form. This has advantages and disadvantages similar to those of the previous alternative.

- The browser extension could store the RP address encrypted in a cookie as part of step 3, so that the browser extension can obtain the address in step 11. In order to adhere to cookie security rules [31], this must be done in such a way that the browser believes it is communicating with the same domain when the cookie is set and when it is retrieved[21].

  To achieve this, the browser extension encrypts and stores the RP address in a cookie in step 3, before the identity selector is invoked. As part of step 8, the browser extension retrieves the encrypted value from the cookie and sends it to the IdP as a hidden HTML variable in an HTML form or as a query URL parameter. As part of step 10, the IdP returns the encrypted RP address to the user agent (again as a hidden form variable or as a URL parameter[22]). In step 11, the browser extension obtains the encrypted value and decrypts it to obtain the RP address.

  Note that the IdP is unable to read the RP address, hence protecting user privacy, since it is encrypted using a key known only to the browser extension. If the IdP, however, needs the RP address for auditing purposes (e.g. for legal reasons), or the IdP policy requires the disclosure of the RP identity (e.g. so it can encrypt the security token using the RP's public key), then the RP address could be sent in plain text to the IdP.

[21]Note that creation of and access to the cookie can be handled by the browser extension transparently to RPs and IdPs.

4.5 Defeating Phishing

Use of LibertyCards helps to mitigate the risk of phishing. The LibertyCard contains the URL of the IdP entered by the user, and the user will only be forwarded to that IdP, i.e. the RP will not be able to redirect the user to an IdP of its choice. By contrast, in the Liberty artifact and Liberty browser post profiles (and in OpenID [44, 48]), a malicious SP might redirect a user to a fake IdP, which could then capture the user credentials. This is a particular threat for static credentials, such as usernames and passwords.

4.6 Integration at the Client Side

Some IdPs and RPs/SPs may not be prepared to accept the burden of supporting two identity management systems simultaneously, at least unless there is a significant financial incentive. Currently, major Internet players, such as MSN[23], do not provide any means of interoperating between identity management systems. As a result, a client-side technique for supporting interoperation could be practically useful.

In addition, building the integration scheme on the client means that the performance of the server is not affected, since the integration overhead is handled by the client. Such an approach also reduces the load on the network.

4.7 STS-enhanced RPs

STS-enhanced RPs are not supported by the integration scheme. This is because use of an STS involves direct communication (i.e. not via a browser) between the CardSpace identity selector and the RP STS [27], which the integration browser extension is currently not capable of intercepting. For example, the identity selector directly contacts the RP STS to obtain its security policy using WS-MetadataExchange.

In the scheme described in this paper, the interaction with the RP uses HTTP/HTML via a web browser. This is a simpler and probably more common scenario for RP interactions [19]. As discussed in section 2.1.3, an RP security policy can be expressed using HTML, and both the policy and the security token can be exchanged using HTTP/S. Therefore, to act as a CardSpace-enabled RP, a website is not

[22]The use of HTML forms (with the POST method) is preferable to query URL parameters, since the latter may suffer from size restrictions; hence the former approach is used in the prototype implementation described in section 5.

[23]`http://www.msn.com`

required to implement any of the WS-* specifications [19, 27].

4.8 Applicability of the Scheme

Although the proposed integration scheme is presented as Liberty-specific, we suspect that the scheme could also be applicable for SAML-compliant IdPs; this, nevertheless, requires certain modifications to the current scheme. For example, the technical differences[24] between Liberty ID-FF 1.2 and SAML 2.0 must be carefully examined. However, given that SAML 2.0 is the successor to SAML 1.1, Liberty ID-FF 1.2 and Shibboleth 1.3 [15], a mapping seems likely to be possible.

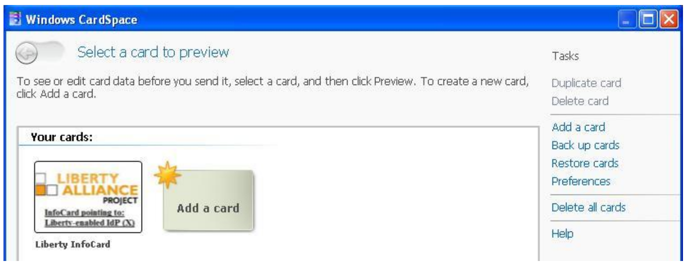
Reconfiguring the integration scheme to interoperate with SAML-aware IdPs potentially significantly increases its applicability and practicality. For example, the exchange of identity attributes, which is not supported under the current scheme, would then be feasible. The reconfiguration of the scheme remains possible future work.

5. PROTOTYPE REALISATION

This section provides technical details of a prototype implementation of the integration scheme when used with the Liberty browser post profile. A number of prototype-specific properties and possible limitations of the current prototype are also described.

5.1 User Registration

Prior to use, the user must have accounts with a CardSpace RP and a Liberty-enabled IdP. The user must also create a LibertyCard for the relevant Liberty IdP (or it could be created at the time of use). This involves invoking the CardSpace identity selector and inserting the URL of the target Liberty IdP in the *web page* field[25] and the trigger word (*Liberty*) in the *city* field. For ease of identification, the user can give the personal card a meaningful name, e.g. of the target IdP site. The user can also upload an image for the card, e.g. containing the logo of the intended IdP or simply of Liberty. When a user wishes to use a particular Liberty IdP, the user simply chooses the corresponding card. An example of a LibertyCard is shown in figure 4.



To see or edit card data before you send it, select a card, and then click Preview. To create a new card, click Add a card.

Figure 4: A LibertyCard

[24]https://spaces.internet2.edu/display/SHIB/SAMLLibertyDiffs

[25]The *web page* field was chosen to contain the Liberty IdP URL since it seems the logical choice; however, this is an implementation option.

5.2 Implementation Details

The prototype, described in section 5.3, was coded as a client-side plug-in[26] using JavaScript [40, 42], chosen to maximise portability. Indeed, JavaScript[27] appears to be the most widely browser-supported and commonly used client-side scripting language across the Web today. Use of browser-specific client-side scripting languages, e.g. VBScript, was ruled out to ensure the widest applicability [20].

The implementation uses the Document Object Model (DOM) [32] to inspect and manipulate HTML [43] pages and XML [9] documents. Since the DOM defines the objects and properties of all document elements and the methods to access them, a client-side scripting language can read and modify the contents of a web page or completely alter its appearance [20].

The prototype does not use any of the published CardSpace application programming interfaces (APIs). This will ease migration of the plug-in to other CardSpace-like systems such as the Linux/Mac-based DigitalMe[28] and the Firefox/Safari InfoCard extensions.

5.3 Operation of the Prototype

In this section we consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 3.2.5.

5.3.1 Prototype-specific Operational Details

In step 3, before the HTML login page is displayed, the plug-in uses the DOM to perform the following processes.

1. The plug-in scans the web page in the following way[29].
   (a) It searches through the HTML elements of the web page to detect whether any HTML forms are present. If so, it searches each form, scanning through each of its child elements for an HTML object tag.
   (b) If an object tag is found, it retrieves and examines its type. If it is of type 'application/x-informationCard' (which signals website support for CardSpace), it continues; otherwise it aborts.
   (c) It then searches through the param tags (child elements of the retrieved CardSpace object tag) for the 'requiredClaims' tag, which lists the claims required by the RP security policy.
   (d) If the required claims include attributes other than the PPID claim, then the plug-in terminates, giving CardSpace the opportunity to operate normally. However, if only the PPID claim is requested, then the plug-in adds the *city* and *web page* claims to the 'requiredClaims' tag, marking them as mandatory (see section 3.2.3).

[26]We use the term *plug-in* to refer to any client-side browser extension, such as a user script, plug-in, etc.

[27]Throughout the description the term *JavaScript* is, for simplicity, used to refer to all variants of the language.

[28]http://code.bandit-project.org/trac/wiki/DigitalMe

[29]The relevant user guide [27] specifies two HTML extension formats for invoking an identity selector from a web page, both of which include placing the CardSpace object tag inside an HTML form. This motivates the choice of the web page search method.

2. The plug-in adds a JavaScript function to the head section of the HTML page to intercept the XML-based authentication token before it is sent back to the RP (such a token will be sent by the identity selector in step 8).

3. The plug-in obtains the current action attribute of the CardSpace HTML form, encrypts it using AES [39] with a secret key known only to the plug-in, and then stores it in a cookie. This attribute specifies the URL address of a web page at the CardSpace-enabled RP to which the authentication token must be forwarded for processing. If the obtained attribute is not a fully qualified domain name address, the JavaScript inherent properties, e.g. *document.location.protocol* and *document.location.host*, are used to help reconstruct the full URL address.

4. After storing it, the plug-in changes the current action attribute of the CardSpace HTML form to point to the newly created 'interception' function (see step 2 above).

5. The plug-in creates and appends an 'invisible' HTML form to the HTML page to be used later for sending the SAML token request to the Liberty-enabled IdP.

In step 8 the plug-in uses the DOM to perform the following steps.

1. It intercepts the RSTR message sent by the CardSpace identity selector using the added function (see above).

2. It parses the intercepted token. If the *city* field contains the word Liberty, the plug-in proceeds; if not, normal operation of CardSpace continues. It also reads the *web page* field to discover the URL address of the IdP. In addition, all other fields, including the PPID and InfoCard public key with its digital signature, are parsed. The *city*, *web page*, and *PPID* fields are contained in a SAML attribute statement, whereas the public key and signature values are contained in a SAML signature statement.

   The plug-in uses an XML parser built into the browser to read and manipulate the intercepted XML token. The plug-in passes the token to the parser, which reads it and converts it into an XML DOM object that can be accessed and manipulated by JavaScript. The DOM views the XML token as a tree-structure, thereby enabling JavaScript to traverse the DOM tree to read (and possibly modify) the content of the token elements. New elements can also be created where necessary.

3. It converts the token format from a SAML response message into a SAML request message, compatible with Liberty-conformant IdPs supporting the browser post profile. This involves converting a SAML 1.1-based RSTR into a SAML 2.0 authentication request. Moreover, as outlined in section 3.2.2, the plug-in adds the PPID and the InfoCard public key along with its signature to the SAML request message, because the token must be signed by the Liberty-enabled IdP to provide integrity and authenticity services.

4. It writes the entire SAML request message as a hidden variable into the invisible HTML form created earlier.

5. It retrieves the encrypted RP URL from the cookie, and writes it into the invisible form as a hidden variable.

6. It writes the URL address of the Liberty IdP into the action attribute of the invisible form.

7. It auto-submits the HTML form (transparently to the user), using the JavaScript method 'click()' on the 'submit' tag.

5.3.2 Liberty IdP-specific Details

For steps 8 to 10, we have created an experimental website to act as a Liberty-enabled IdP supporting the Liberty browser post profile. PHP is used to enable the IdP to parse the SAML request and perform the user authentication. The user credentials, i.e. username and password, that the IdP uses to authenticate the user are stored in a MySQL database. They are salted, hashed with SHA-1, and protected against SQL injection attacks. PHP supports a variety of XML parsers, such as XML DOM, Expat parser, and SimpleXML. The prototype uses XML DOM.

5.3.3 User Consent and Token Forwarding

In step 11, the plug-in operates as follows.

1. It obtains the encrypted value of the RP URL from the appropriate HTML hidden variable, decrypts it using its internally stored secret key, and inserts it into the action attribute of the HTML form carrying the received SAML token.

2. The plug-in then displays the token to the user and requests consent to proceed. The displayed token indicates the types of information the authentication token is carrying, as well as the exact URL address of the RP to which the token will be forwarded. The JavaScript 'confirm()' pop-up box is used to achieve this.

3. If the user approves the token, the plug-in seamlessly submits it to the RP using the JavaScript 'click()' method.

5.3.4 CardSpace RP-specific Details

To test the prototype, we built an experimental website to act as a CardSpace-enabled RP. On receipt of the SAML authentication token, the RP uses PHP in step 11 to parse and validate the received token. As is the case with the Liberty IdP, the user identifying data is salted, hashed and stored in a MySQL database that is resistant to SQL injection attacks. The validation process includes verifying the digital signatures and checking the conditions, e.g. time stamps, included in the token. The PPID and the InfoCard public key in the token are compared to the values stored in the RP database, and the authentication status is also checked.

5.3.5 Other Issues

The JavaScript-driven plug-in was built using IE7PRO, an IE extension, chosen to expedite the prototype implementation. Users of the prototype must therefore install IE7PRO, freely available at the IE7PRO website[30], prior to installing the integration plug-in. To enable or disable

[30]`http://www.ie7pro.com`

the integration prototype, a user can simply tick or un-tick the appropriate entry in the 'IE7PRO Preferences' interface. This provides the means to achieve the final objective listed in section 3.2.4.

Finally note that the integration plug-in does not require any changes to default IE security settings, thereby avoiding potential vulnerabilities resulting from lowering browser security settings.

5.4 Limitations

The current version of the prototype has not been tested with CardSpace relying parties using TLS/SSL. Therefore, we are not able to provide precise operational and performance details in this case.

If the RP has a certificate, then the identity selector will, by default, encrypt the SAML-based RSTR message using the public key of the requesting RP. Clearly, the plug-in does not have access to the RP's private key, and hence will not be able to decrypt the token. Therefore, it will not know whether to trigger the integration protocol, and will be unable both to discover which IdP it must contact, and to obtain the user identifier (the PPID).

One solution to these issues would be for the plug-in to first ask the user whether the integration protocol should be activated (e.g. via a JavaScript prompt window), and, if so, it should then forward the SAML token to the RP and notify the RP to wait for another token. The RP should decrypt the token, read the PPID, and then wait. At the same time, the plug-in should prompt the user to enter the URL of the Liberty-enabled IdP, and then create and send a SAML request message to the Liberty IdP, which authenticates the user and responds with a SAML response token. The plug-in could then, optionally, seek user consent, and, if the user approves, the plug-in would then forward the token to the RP. The RP must issue the plug-in with a nonce (and a time-stamp) which the plug-in sends back with the second token to both link the two tokens together and help protect against replay and guessing attacks.

One of the most obvious drawbacks to this solution is that it requires changes at the CardSpace-enabled RP, as the RP must be reconfigured to accept two tokens. However, this would not be a major change since both tokens will be constructed using SAML, and since the RP is not required to directly contact the Liberty-enabled IdP. Therefore, the major overheard remains with the client. Nevertheless, we are working on a revised version of the prototype that is fully compatible with SSL/TLS encryption but without the requirement of RP reconfiguration.

The integration plug-in must scan every browser-rendered web page to detect whether it supports CardSpace, and this may affect system performance. However, informal tests on the prototype suggest that this is not a serious issue. In addition, the plug-in can be configured so that it only operates with certain websites.

The integration plug-in has not been tested with CardSpace 2.0, because it was completed well before its release. Therefore, we are not yet able to provide precise operational details for this version.

Finally note that some older browsers (or browsers with scripting disabled) may not be able to run the integration plug-in, as it was built using JavaScript. However, most modern browsers support JavaScript (or ECMAscript), and hence building the prototype in JavaScript is not a major usability obstacle.

6. RELATED WORK

The Bandit[31] and Concordia[32] projects are currently developing open source technologies to support interoperation between identity management systems. Unlike the integration scheme proposed in this paper, these systems are not based on client-side models. Concordia has proposed a CardSpace and SAML/WS-Federation integration model. This could be used as the basis for supporting Liberty/CardSpace interoperation by taking advantage of the similarities between the Liberty ID-FF SSO profiles and the SAML SSO profiles.

Another scheme supporting interoperation between CardSpace and Liberty has been proposed by Jørstad et al. [29]. In this scheme, the IdP is responsible for supporting interoperation. The IdP must therefore perform the potentially onerous task of maintaining two different identity management schemes. In addition, this scheme requires the user to possess a mobile phone supporting the Short Message Service (SMS). Moreover, the IdP must always perform the same user authentication technique, regardless of the identity management system the user is attempting to use. The IdP simply sends an SMS to the user, and, in order to be authenticated, the user must confirm receipt of the SMS. This confirmation is also an implicit user approval for the IdP to send a security token to the RP. By contrast, the scheme proposed in this paper does not require use of a handheld device, and does not enforce a specific authentication method.

Finally, we observe that Liberty is apparently also working on a scheme somewhat similar to that described here. No specifications have yet been released, but the plans are described in a presentation available at the Liberty website[33].

7. CONCLUSIONS AND FUTURE WORK

We have proposed a means of interoperation between two leading identity management systems, namely CardSpace and Liberty. CardSpace users are able to obtain an assertion token from a Liberty-enabled identity provider that satisfies the security requirements of a CardSpace-enabled relying party. The scheme uses a client-side browser extension, and requires no major changes to servers. It uses the CardSpace identity selector interface to integrate Liberty identity providers with CardSpace relying parties. The scheme extends the use of personal cards to allow for such interoperability.

The integration scheme takes advantage of the similarity between the Liberty ID-FF and the CardSpace frameworks, and this should help to reduce the effort required for full system integration. Also, implementation of the scheme does not require technical co-operation between Microsoft and Liberty.

Planned future work includes investigating the possibility of using the CardSpace identity selector to enable access to identity providers of other identity management systems, such as OpenID and Shibboleth. In addition, we also plan to

[31]`http://www.bandit-project.org`

[32]`http://www.projectconcordia.org`

[33]`http://www.projectliberty.org/liberty/content/download/4541/31033/file/20080ICP-Cardspace-DIDW.pdf`

investigate the possibility of extending the proposed integration protocol to support CardSpace-enabled relying parties that employ security token services.

8. REFERENCES

[1] W. A. Alrodhan and C. J. Mitchell. A client-side CardSpace-Liberty integration architecture. In *Proceedings of the 7th Symposium on Identity and Trust on the Internet (IDtrust 08)*, pages 1–7. ACM, New York, NY, USA, 2008.

[2] S. Anderson et al. *Web Services Trust Language (WS-Trust).* Actional Corporation, BEA Systems, Computer Associates International, International Business Machines Corporation, Layer 7 Technologies, Microsoft Corporation, Oblix, OpenNetwork Technologies, Ping Identity Corporation, Reactivity, RSA Security, and VeriSign, 2005. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-trust/ws-trust.pdf.

[3] S. Bajaj et al. *Web Services Policy Framework (WS-Policy).* BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Sonic Software, and VeriSign, 2006. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf.

[4] K. Ballinger et al. *Web Services Metadata Exchange (WS-MetadataExchange).* BEA Systems, Computer Associates International, International Business Machines Corporation, Microsoft Corporation, SAP AG, Sun Microsystems, and webMethods, 2006. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-mex/metadataexchange.pdf.

[5] A. Berger. *Identity Management Systems — Introducing Yourself to the Internet.* VDM Verlag, Saarbrücken, Germany, 2008.

[6] V. Bertocci, G. Serack, and C. Baker. *Understanding Windows CardSpace: An Introduction to the Concepts and Challenges of Digital Identities.* Addison-Wesley, Reading, Massachusetts, USA, 2008.

[7] K. Bhargavan, C. Fournet, A. D. Gordon, and N. Swamy. Verified implementations of the information card federated identity-management protocol. In *Proceedings of the 2008 ACM symposium on Information, Computer and Communications Security (ASIACCS 08)*, pages 123–135. ACM, New York, NY, USA, 2008.

[8] D. Birch. *Digital Identity Management: Technological, Business and Social Implications.* Gower Publishing, Farnham, UK, 2007.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau (editors). *Extensible Markup Language (XML) 1.0.* W3C Recommendation, 5th edition, 2008. http://www.w3.org/TR/xml/.

[10] K. Cameron. *The Laws of Identity.* Microsoft Corporation, 2005. http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf.

[11] K. Cameron and M. B. Jones. *Design Rationale behind the Identity Metasystem Architecture.* Microsoft Corporation, 2006. http://www.identityblog.com/wp-content/resources/design_rationale.pdf.

[12] S. Cantor, J. Kemp, and D. Champagne (editors). *Liberty ID-FF Bindings and Profiles Specification.* Liberty Alliance Project, 2004. http://www.projectliberty.org/liberty/content/download/319/2369/file/draft-liberty-idff-bindings-profiles-1.2-errata-v2.0.pdf.

[13] S. Cantor, J. Kemp, R. Philpott, and E. Maler (editors). *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.* OASIS, 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[14] S. Cantor and J. Kemp (editors). *Liberty ID-FF Protocols and Schema Specification.* Liberty Alliance Project, 2005. http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_ff_1_2_specifications.

[15] S. Cantor (editor). *Shibboleth Architecture — Protocols and Profiles*, 2005. http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-protocols-200509.pdf.

[16] D. Chadwick. FileSpace: an alternative to CardSpace that supports multiple token authorisation and portability between devices. In *Proceedings of the 8th Symposium on Identity and Trust on the Internet (IDtrust 09)*, pages 94–102. ACM, New York, NY, USA, 2009.

[17] D. W. Chadwick. Federated identity management. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 96–120. Springer, Berlin/Heidelberg, Germany, 2009.

[18] D. W. Chadwick and G. Inman. Attribute aggregation in federated identity management. *IEEE Computer*, 42(5):33–40, 2009.

[19] D. Chappell. *Introducing Windows CardSpace.* MSDN, 2006. http://msdn.microsoft.com/en-us/library/aa480189.aspx.

[20] N. Daswani, C. Kern, and A. Kesavan. *Foundations of Security: What Every Programmer Needs to Know.* Apress, Berkeley, CA, USA, 2007.

[21] G. Della-Libera et al. *Web Services Security Policy Language (WS-Security Policy).* International Business Machines Corporation, Microsoft Corporation, RSA Security, and VeriSign, 2005. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf.

[22] R. Fielding, J. Getty, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol — HTTP/1.1.* RFC 2616, The Internet Society, 1999. http://tools.ietf.org/html/rfc2616.

[23] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* RFC 2045, Internet Engineering Task Force, 1996. http://www.ietf.org/rfc/rfc2045.txt.

[24] S. Gajek, J. Schwenk, M. Steiner, and C. Xuan. Risks of the CardSpace protocol. In *Proceedings of the 12th International Conference on Information Security (ISC 09)*, pages 278–293. Springer-Verlag, Berlin/Heidelberg, Germany, 2009.

[25] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon (editors). *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation, 2007. `http://www.w3.org/TR/soap12-part1/`.

[26] P. Hallam-Baker and E. Maler (editors). *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.0*. OASIS, 2002. `http://www.oasis-open.org/specs/#samlv1.0`.

[27] M. B. Jones. *A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers*. Microsoft Corporation, 2008.

[28] M. B. Jones and M. McIntosh (editors). *Identity Metasystem Interoperability Version 1.0 (IMI 1.0)*. OASIS Standard, 2009. `http://docs.oasis-open.org/imi/identity/v1.0/identity.html`.

[29] I. Jørstad, D. Van Thuan, T. Jønvik, and D. Van Thanh. Bridging CardSpace and Liberty Alliance with SIM authentication. In *Proceedings of the 10th International Conference on Intelligence in Next Generation Networks (ICIN 07)*, pages 8–13. Adera, BP 196 - 33608 Pessac Cedex, France, 2007.

[30] S. Kellomäki and R. Lockhart (editors). *Liberty ID-SIS Employee Profile Service Specification*. Liberty Alliance Project, 2005. `http://www.projectliberty.org/liberty/content/download/1031/7155/file/liberty-idsis-ep-v1.1.pdf`.

[31] D. Kristol. *HTTP State Management Mechanism*. RFC 2045, Internet Engineering Task Force, 2000. `http://tools.ietf.org/html/rfc2965`.

[32] A. Le Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne (editors). *Document Object Model (DOM) Level 2 Core Specification*. W3C Recommendation, 2000. `http://www.w3.org/TR/DOM-Level-2-Core/`.

[33] E. Maler, P. Mishra, and R. Philpott (editors). *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*. OASIS, 2003. `http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf`.

[34] M. Mercuri. *Beginning Information Cards and CardSpace: From Novice to Professional*. Apress, New York, USA, 2007.

[35] Microsoft Corporation. *Microsoft's Vision for an Identity Metasystem*, May 2005. `http://msdn.microsoft.com/en-us/library/ms996422.aspx`.

[36] Microsoft Corporation and Ping Identity Corporation. *An Implementer's Guide to the Identity Selector Interoperability Profile v1.5*, 2008.

[37] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker (editors). *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. OASIS Standard Specification, 2006. `http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`.

[38] A. Nanda and M. B. Jones. *Identity Selector Interoperability Profile V1.5*. Microsoft Corporation, 2008. `http://www.identityblog.com/wp-content/resources/2008/Identity_Selector_Interoperability_Profile_V1.5.pdf`.

[39] National Institute of Standards and Technology (NIST). *Announcing the Advanced Encryption Standard (AES)*, FIPS 197, November 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[40] T. Negrino and D. Smith. *JavaScript and Ajax for the Web: Visual QuickStart Guide*. Peachpit Press, Berkeley, CA, USA, 7th edition, 2008.

[41] R. Oppliger, S. Gajek, and R. Hauser. Security of Microsoft's identity metasystem and CardSpace. In *Proceedings of the Kommunikation in Verteilten Systemen (KiVS 07)*, pages 63–74. VDE Publishing House, Berlin, Germany, 2007.

[42] T. A. Powell and F. Schneider. *Javascript: The Complete Reference*. McGraw-Hill Osborne Media, Berkeley, CA, USA, 2nd edition, 2004.

[43] D. Raggett, A. L. Hors, and I. Jacobs (editors). *HTML 4.01 Specification*. W3C Recommendation, 1999. `http://www.w3.org/TR/html401/`.

[44] D. Recordon, L. Rae, and C. Messina. *OpenID: The Definitive Guide*. O'Reilly Media, Sebastopol, CA, USA, 2010.

[45] T. Scavo (editor). *SAML V2.0 Holder-of-Key Assertion Profile Version 1.0*. OASIS, 2009. `http://www.oasis-open.org/committees/download.php/34962/sstc-saml2-holder-of-key-cd-03.pdf`.

[46] D. Todorov. *Mechanics of User Identification and Authentication: Fundamentals of Identity Management*. Auerbach Publications, New York, USA, 2007.

[47] J. Tourzan and Y. Koga (editors). *Liberty ID-WSF Web Services Framework Overview*. Liberty Alliance Project, 2005. `http://www.projectliberty.org/liberty/content/download/1307/8286/file/liberty-idwsf-overview-v1.1.pdf`.

[48] R. Ur Rehman. *Get Ready for OpenID*. Conformix Technologies, Chesterbrook, Pennsylvania, USA, 2008.

[49] T. Wason (editor). *Liberty ID-FF Architecture Overview*. Liberty Alliance Project, 2003. `http://www.telenor.com/rd/idm/liberty-idff-arch-overview-v1.2.pdf`.

[50] G. Williamson, D. Yip, I. Sharoni, and K. Spaulding. *Identity Management: A Primer*. MC Press, Big Sandy, TX, USA, 2009.