

# Browser Extension-based Interoperation Between OAuth and Information Card-based Systems

Haitham S. Al-Sinani

Technical Report  
RHUL-MA-2011-15  
24 September 2011



Department of Mathematics  
Royal Holloway, University of London  
Egham, Surrey TW20 0EX, England

<http://www.rhul.ac.uk/mathematics/techreports>

## 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 simple scheme to provide client-based interoperation between OAuth and an Information Card-based system such as CardSpace or Higgins. In this scheme, Information Card users are able to obtain an assertion token from an OAuth-enabled system, the contents of which can be processed by an Information Card-enabled relying party. The scheme, based on a browser extension, is transparent to OAuth providers and to identity selectors, and only requires minor changes to the operation of an Information Card-enabled relying party. We specify its operation and also describe an implementation of a proof-of-concept prototype. Additionally, security and operational analyses are provided.

**Keywords:** Information Cards, CardSpace, Higgins, OAuth, Interoperation, Browser Extension

## 1 Introduction

In an attempt to simplify management of identities and mitigate identity-oriented attacks, a number of identity management systems (e.g. CardSpace, OAuth, OpenID, etc.) have been proposed [4]. An identity provider (IdP) in such a system supplies a user agent (UA), typically a web browser, with an assertion token that can be consumed by a particular relying party (RP). Whilst one RP might solely support an Information Card system, another might only support OAuth. Therefore, to make these systems available to the largest possible group of users, effective interoperability between such systems is needed. In this paper we investigate a case involving an Information Card-enabled RP, an OAuth-enabled provider, and a UA that supports Information Cards. The goal is to develop a client-based approach to integration that is as transparent as possible to IdPs, RPs and identity selectors.

The scheme operates with a variety of Information Card-based systems, including CardSpace and Higgins. For simplicity of presentation, in this paper we describe its operation with CardSpace, a widely-discussed example of an Information Card-based system.

We consider CardSpace-OAuth interoperation because of OAuth's fast-growing adoption by widely used Internet service providers such as Facebook and Twitter. Complementing this, the wide use of Windows, recent

versions of which incorporate CardSpace, means that enabling interoperability between the two systems is likely to be of significance for large numbers of identity management users and service providers. CardSpace-OAuth interoperability is also attractive since both schemes support the exchange of user attributes.

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

## 2 CardSpace and OAuth

### 2.1 CardSpace

#### 2.1.1 Introduction

CardSpace provides a secure and consistent way for users to control and manage personal data, to review personal data before sending it to a website, and to verify the identity of visited websites. It also enables websites to obtain data from users, e.g. to support user authentication and authorisation.

Digital identities are represented to users as Information Cards (or InfoCards). There are two types of InfoCards: personal (self-issued) cards and managed cards, issued by remote IdPs. Personal cards are created by users themselves, and the claims listed in such an InfoCard are asserted by the self-issued identity provider (SIIP) that co-exists with the CardSpace identity selector (or just the selector) on the user machine. InfoCards do not contain sensitive information, but instead carry metadata indicating the types of personal data 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 IdP that issued it [5, 13].

By default, CardSpace is supported by Internet Explorer (IE) from version 7 onwards. Extensions to other browsers, such as Firefox<sup>1</sup> and Safari<sup>2</sup>, also exist. An updated version, CardSpace 2.0 Beta 2, was released, although Microsoft announced in early 2011 that it will not ship; instead

---

<sup>1</sup><https://addons.mozilla.org/en-US/firefox/addon/cardspace-support-for-firefox/>

<sup>2</sup><http://www.hccp.org/safari-plug-in.html>

Microsoft has released a technology preview of U-Prove<sup>3</sup>. In this paper we refer throughout to the CardSpace version that is shipped by default as part of Windows Vista and Windows 7, that is available as a free download for XP and Server 2003, and which has been approved as an OASIS standard [11].

### 2.1.2 Personal Cards

The scheme proposed here uses CardSpace personal cards to make information provided by an OAuth IdP (Resource Server) available to CardSpace RPs via the selector. The selector allows a user to create a personal card and populate its fields with self-asserted claims. CardSpace restricts the contents of personal cards to non-sensitive data in the form of 14 editable claim types including *First Name*, *Last Name*, *Web Page* and *Email Address*. Data inserted in personal cards is stored in encrypted form on the user machine. At the time of creation, a card ID and a card master key are created and stored.

**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), a service responsible for token management [10].

1. UA  $\rightarrow$  RP. HTTP/S request: GET (login page). A user visits a CardSpace-enabled RP login page.
2. RP  $\rightarrow$  UA. HTTP/S response. A login page is returned containing the CardSpace-enabling tags in which the RP security policy is embedded.
3. User  $\rightarrow$  UA. The RP page offers the option to use CardSpace; selecting this option activates the selector, which is passed the RP policy. If this is the first time that this RP has been contacted, the selector will display the identity of the RP and give the user the option to either proceed or abort the protocol.
4. Selector  $\rightarrow$  InfoCards. The selector, after evaluating the RP policy, highlights InfoCards matching the policy and greys out the rest. InfoCards previously used for this RP are displayed in the upper half of the selector screen.

---

<sup>3</sup><http://blogs.msdn.com/b/card/archive/2011/02/15/beyond-windows-cardspace.aspx>

5. User  $\rightarrow$  InfoCards. The user chooses a personal card. (Alternatively, the user could create and choose a new personal card). The user can preview the card (with its associated claims) to ensure that they are willing to release the claim values. Of the claims specified in an InfoCard, only those requested in the RP policy will be passed to the requesting RP.
6. Selector  $\rightleftharpoons$  SIIP. The selector creates and sends a SAML-based Request Security Token (RST) to the SIIP, which responds with a SAML-based Request Security Token Response (RSTR).
7. UA  $\rightarrow$  RP. The RSTR is passed to the UA, which forwards it to the RP.
8. RP  $\rightarrow$  UA. The RP verifies the token, and, if satisfied, grants access.

**Private Personal Identifiers (PPIDs)** The PPID is an identifier linking a specific InfoCard to a particular RP [5]. When a user first uses a personal card at a particular RP, CardSpace generates both a card-site-specific PPID by combining the card ID with data taken from the RP certificate, and a card-site-specific signature key pair as a function of the card master key and data taken from the RP certificate. The RP domain/IP address 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 and reduce the impact of PPID compromise. The selector displays a shortened version of the PPID to protect against social engineering attacks and improve readability.

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

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 security token to provide a more robust authentication method [5].

## 2.2 OAuth

### 2.2.1 Introduction

OAuth<sup>4</sup> (Open Authorisation) is an emerging, open, identity management standard, enabling an end-user to grant an Internet application controlled access to personal information (e.g. user attributes, photos, contact lists, etc.) stored at a third party site, without divulging long-term credentials such as passwords. In the absence of a system like OAuth, applications must request user credentials in order to access user information held by a third party, which is clearly undesirable.

The four entities involved in the OAuth protocol are: the *Resource Owner*, typically an end-user (or, more specifically, their UA); the *Client*, an application requesting access to user resources; the *Resource Server*, a server hosting user resources; and the *Authorisation Server*, a server that issues access tokens to clients after successfully authenticating the Resource Owner and obtaining its authorisation. Note that the latter two roles are typically performed by a single entity.

### 2.2.2 Operational Protocol

Two major (incompatible) versions of OAuth have been released: OAuth 1.0 [6] and 2.0 [8, 15]. We next describe the latest version, OAuth 2.0.

The OAuth protocol enables a Client to request authorisation from the Resource Owner for access to specific information held by a Resource Server, possibly via an intermediary Authorisation Server (the latter option is recommended [8]). If necessary, the Authorisation Server first authenticates the Resource Owner and, if successful, asks the Resource Owner to authorise the Client. If the Resource Owner decides to grant this request, an authorisation token (known as an authorisation grant) is sent to the Client (four authorisation grant types are defined — see below). The Client then requests an access token<sup>5</sup> from the Authorisation Server, where the request includes the authorisation grant. The Authorisation Server authenticates the Client and verifies the authorisation grant, and, if successful, issues an access token. Next, the Client requests access to the private resource(s) from the Resource Server, presenting the access token. Finally, the Resource Server

---

<sup>4</sup><http://oauth.net/>

<sup>5</sup>An access token is typically an opaque string that indicates permission to access specific information for a limited time period; such a token can be independently revoked. The access token must be kept confidential, and should be issued with the minimum necessary scope and lifetime.

verifies the access token, and, if it is valid, meets the request.

The four authorisation types supported by OAuth 2.0 are: authorisation code, implicit, resource owner password credentials, and client credentials, corresponding to four possible protocol flows. The two types of most relevance here are discussed below.

**Authorisation Code.** An authorisation code is typically a short-lived random string. Such a value is supplied by an Authorisation Server to a Client if a Resource Owner authorises a request made by the Client to access (a) specific user resource(s).

The Client redirects the Resource Owner UA to the Authorisation Server, requesting access to personal data. If necessary, the Authorisation Server authenticates the Resource Owner; if successful, the Authorisation Server asks the Resource Owner to authorise the Client request, and, if the Resource Owner agrees, the Authorisation Server issues an authorisation code to the Client. The Client uses the authorisation code to request an access token from the Authorisation Server. Before issuing such a token, the Authorisation Server first authenticates the Client by checking that the credentials provided by the Client match those issued to it when it registered with the Authorisation Server. If successful, the Authorisation Server generates an access token and sends it to the Client via a secure back-channel. The use of this back channel means that this grant type cannot be supported by the scheme proposed here.

**Implicit.** An authorisation grant is said to be ‘implicit’ if the access token is issued to the Client as a direct result of Resource Owner authorisation. Since it requires fewer round trips to obtain an access token than for the authorisation code type, the implicit grant type improves the responsiveness and efficiency of certain clients, including browser-hosted client applications. Such a grant type is supported by the scheme proposed in this paper.

Before use of the OAuth operational protocol, the Client must register with the OAuth Authorisation Server. How this is achieved is beyond the scope of the OAuth specifications [8], but it could involve the use of an HTML registration form provided by the Authorisation Server. During registration, the Authorisation Server collects certain data about the Client, including the Client type, its redirection URI, and any other server-required data, e.g. the Client name. The Authorisation Server issues the registered

Client with a unique identifier and a secret used for Client authentication when using the authorisation code grant type.

In the remainder of this paper we restrict our attention to the OAuth 2.0 protocol when using the implicit grant type. In this case the OAuth protocol operates as follows.

1. Client  $\rightarrow$  UA  $\rightarrow$  Authorisation Server: HTTP Request. The Client redirects the Resource Owner to the Authorisation Server, requesting access to private data. The redirect includes the Client identifier, the scope of the requested access, an optional state parameter, and a redirection URI to which the Authorisation Server will redirect the Owner once access is granted (or denied). The optional (but recommended) state parameter is set equal to an unguessable value [8]. It is used by the Client to match its initial redirection to the response from the Authorisation Server.
2. Authorisation Server  $\rightleftharpoons$  Resource Owner (UA): Verification and Authorisation. The Authorisation Server validates the received HTTP request (see step 1), ensuring that all the required parameters are present and valid. The Authorisation Server verifies the Client identity by comparing the provided redirection URI with the Client URI previously registered. If the request is valid and the Client identity is successfully verified, the Authorisation Server (if necessary) authenticates the Resource Owner (via the UA) over a TLS-protected channel; the authentication method used is outside the scope of OAuth. It then asks the Resource Owner whether the Client's access request is authorised (where this query includes the Client identifier and the scope of the requested access). If all the checks succeed, the protocol continues.
3. Authorisation Server  $\rightarrow$  UA  $\rightarrow$  Client: HTTP Response. The Authorisation Server redirects the Resource Owner's UA back to the Client using the redirection URI provided earlier (see step 1). The redirection URI includes the access token in a URI fragment. If the state parameter was present in step 1, the Authorisation Server must return it unmodified (in the URI) to the Client to protect against attacks involving manipulated URIs and malicious redirections, notably CSRF<sup>6</sup> (Cross-Site Request Forgery) attacks [7]. The UA follows the redirection instructions by making a request to the Client, excluding the access token-bearing fragment, although the UA retains the fragment information locally.

---

<sup>6</sup>[http://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery](http://en.wikipedia.org/wiki/Cross-site_request_forgery)

4. Client  $\rightleftharpoons$  UA. The Client returns a web page containing an embedded script capable of accessing the full redirection URI, including the fragment retained by the UA (see step 3). The UA executes the embedded script, which extracts the access token from the URI fragment and passes it to the Client over a secure channel.
5. Client  $\rightleftharpoons$  Resource Server. Finally, the Client uses the access token to retrieve the required resource(s) from the Resource Server via a TLS-protected channel.

### 2.2.3 Facebook Connect

Facebook Connect<sup>7</sup> [14] implements the OAuth 2.0 standard, providing a single sign-on service. Facebook Connect allows users to sign on to applications (e.g. Facebook-affiliated websites) using their Facebook account, and also enables such applications to access Facebook-hosted user data, subject to user authorisation.

## 3 The Integration Scheme

We now describe the novel scheme. The parties involved are a CardSpace-enabled RP, a CardSpace-enabled UA (e.g. a suitable web browser), a browser extension implementing the protocol described in section 3.2, and an OAuth Resource and Authorisation server; for simplicity, we assume that the roles of both the Resource and the Authorisation Servers are performed by a single entity, which we refer to as the ‘OAuth IdP’.

The browser extension performs the functions of a Client. It obtains an access token from the Authorisation Server and uses this to obtain user attributes from the Resource Server.

### 3.1 Requirements

The scheme has the following operational requirements.

- The user must have an existing relationship with both a CardSpace RP and an OAuth IdP (and so the IdP will have a means of authenticating the user).

---

<sup>7</sup><http://developers.facebook.com/docs/authentication/>

- The user must register the RP with the IdP, in a user-specific manner. This involves the user interacting (via the UA) with an HTML registration page hosted by the IdP, and using this page to send the IdP the RP’s name, URI, and (optionally) locale. The IdP then issues an RP identifier, where the ‘identifier’ is used by the browser extension to identify the RP to the IdP.
- Prior to, or during, use of the integration protocol, the user must create a CardSpace personal card, referred to here as an OAuthCard. This OAuthCard must contain the following data items in specific fields (the choice of which is implementation-specific): the URI of the IdP; the RP’s identifier (as issued by the IdP); and a predefined sequence of characters (e.g. ‘OAuth’), used to trigger the browser extension.
- The RP must not employ an STS. Instead, the RP must express its security policy using HTML/XHTML, and interactions between the selector and the RP must be based on HTTP/S via a web browser (a simpler and probably more common scenario for selector-RP interactions). This is because the scheme uses a browser extension, and is thus incapable of managing the necessary communications with an STS.
- The RP must be prepared to accept an unsigned ‘CardSpace-like’ SAML security token (generated by the browser extension) which includes both the OAuth IdP-supplied attributes and the digitally-signed SIIP-issued RSTR containing the card-RP-specific PPID.

### 3.2 Protocol Operation

The protocol operates as follows (a summary of the protocol is shown in figure 1). Steps 1, 2, and 4–7 are the same as steps 1, 2, and 3–6, respectively, of the personal card protocol given in section 2.1.2.

3. Browser Extension → UA. The extension performs the following steps.
  - (a) It scans the login page to detect whether the RP website supports CardSpace. If so, it proceeds; otherwise it terminates.
  - (b) It examines the RP policy to check whether the use of personal cards is acceptable. If so, it proceeds; otherwise it terminates, giving CardSpace the opportunity to operate normally.
  - (c) It keeps a local copy of any RP-requested claims.

- (d) It determines the communication protocol (HTTP or HTTPS) in use with the RP<sup>8</sup>.
  - (e) If HTTP is in use, it modifies the RP policy to include the types of claim employed in the OAuthCard. For example, if the URI of the IdP is stored in the *web page* field of the OAuthCard, then it must ensure that the RP security policy includes the *web page* claim. Note that adding the claim types to the RP policy ensures that the token supplied by the SIIP 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.
8. Selector→ Browser Extension/UA. Following the user submission of a suitable OAuthCard, the RSTR, unlike in the ‘standard’ case, does not reach the RP; instead the extension intercepts it and temporarily stores it.

If the RP uses HTTP, the extension uses the contents of the RSTR to construct an OAuth request which it forwards to the appropriate IdP, having discovered its address from the RSTR.

If the RP uses HTTPS, the browser extension first asks the user whether the use of the integration protocol is required. If not, it terminates, thereby allowing CardSpace to operate normally. If so, the extension prompts the user to enter the URI of the IdP and the RP’s identifier. The browser extension could offer the user the option to store the supplied values for future logins at this RP. Precisely as in the HTTP case, the extension then constructs an OAuth request.

Note that, in both cases (i.e. HTTP and HTTPS), the ‘implicit’ grant type is adopted. Note also that in both cases the OAuth request includes: the ‘redirect\_uri’ parameter, to which the IdP must later redirect the UA; the ‘scope’ parameter, showing the scope requested<sup>9</sup>; and the ‘state’ parameter.

---

<sup>8</sup>Note that the protocol operates slightly differently depending on whether the RP uses HTTP or HTTPS. This is because, if HTTPS is used, the selector will encrypt the RSTR message using the site’s public key, and the browser extension does not have access to the corresponding private key. Hence, it will not know whether to trigger the integration protocol, and will be unable to obtain the OAuth IdP URI and the RP’s identifier; such issues do not occur if HTTP is used, since the selector will not encrypt the RSTR.

<sup>9</sup>This scope parameter indicates the RP-requested user attribute types (if any) which are to be provided by the OAuth IdP. The browser extension will know what they are since they were stored by it in step 3c. The ‘scope’ parameter helps the IdP determine the scope of the access request; the IdP must ask the user to authorise the release of the requested attribute values.

9. OAuth IdP  $\rightleftharpoons$  User. This step is the same as step 2 of the OAuth 2.0 protocol (implicit grant type) given in section 2.2.2.
10. OAuth IdP  $\rightleftharpoons$  Browser Extension/UA. The IdP redirects the UA back to the provided RP URI, including the access token (in the URI fragment) and the ‘state’ parameter<sup>10</sup>. The extension reads and uses the provided access token to request and retrieve the RP-required user attribute values from the IdP via online communication with the IdP. Note that the UA-IdP communication channel is TLS-protected.
11. Browser Extension/UA  $\rightarrow$  RP. Having retrieved the required user attribute values from the IdP, the browser extension constructs a ‘CardSpace-like’ SAML token and submits it to the RP. Such a token includes the IdP-supplied user attributes and the digitally-signed, SIIP-issued RSTR (which contains the PPID), allowing the RP to verify the SIIP signature (see also sections 4.2 and 4.3).
12. RP  $\rightarrow$  User. The RP verifies the SAML token (including verifying the RSTR signature, PPID, nonce, time-stamps, etc.), and, if satisfied, grants access.

## 4 Discussion and Analysis

### 4.1 Defeating Phishing

The scheme mitigates the risk of phishing. This is because the redirect to the OAuth IdP is initiated by the browser extension and not by the RP, i.e. the RP cannot redirect the user to an OAuth IdP of its choosing. By contrast, in OAuth as it is typically used a malicious site could redirect a user to a fake IdP, which might capture user credentials.

### 4.2 OAuth IdP User Authentication

The SAML token created by the browser extension in step 11 of section 3.2 could be extended to contain an additional field to indicate that the user has been authenticated by a specified OAuth IdP (as well as when and how). Of course, the RP would need to be modified to be able to process such an extra field, although this is likely to be relatively straightforward.

---

<sup>10</sup>The browser extension checks that the value in the state parameter is the same as the one it generated in step 8, is sufficiently current, and has not been previously used. The extension then adds the received value to a list for use in future verifications.

## 4.3 Security Considerations

### 4.3.1 Properties

The unsigned SAML token generated by the browser extension in step 11 of section 3.2 (referred to here as the ‘user token’) includes the PPID, the user attributes as provided by the OAuth IdP, and the digitally-signed, SIIP-issued, RSTR. The RP compares the SIIP-asserted PPID (and the public key) in the user token with its stored values and verifies the digital signature (see section 2.1.2). The RP can thus authenticate the user, link the user to his/her account, and consume the OAuth IdP-supplied attributes, e.g. for authorisation purposes. If the RSTR also contains self-issued attributes, the RP could compare these (locally-stored) attributes with the (remotely-stored) IdP-provided attributes; such a procedure could give the RP added guarantees about the accuracy of these attributes.

It is infeasible for a malicious entity to fabricate a user token to masquerade as a legitimate party since it will not have access to the PPID and the private key necessary to sign the RSTR, both of which are only available if the appropriate InfoCard is selected on the correct platform.

Note that, in protocol step 4, the selector identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this RP for the first time, CardSpace requests the user’s permission to proceed (see section 2.1.2). This helps to support mutual authentication since the user and the RP are both identified to each other.

Finally note that the scheme allows the user attributes to be stored remotely at the OAuth IdP; this has potential security advantages over storing the attributes locally on the user machine, as is currently the case with CardSpace SIIP-issued attributes.

### 4.3.2 Concerns

If the web browser is compromised, then an adversary could steal the user token and use it to impersonate the user. Moreover, if the RP does not use HTTPS, then the SIIP-issued RSTR will not be encrypted. Also, if we assume that the web browser is not a secure environment, it may be possible for a malicious plug-in or other malware to get access to sensitive information present in the (plaintext) RSTR, the plug-in-generated SAML token, or the OAuth IdP-issued access token. However, the same risks apply when manually entering credentials (e.g. username-password) into a browser [9].

### 4.3.3 Additional Security Properties

In certain circumstances, the RP can gain additional assurance in the identity of the user through use of the scheme proposed in section 3. If the RP trusts that the correct browser extension is running unmodified on the user platform, then the RP will know that the user has been authenticated by the OAuth IdP, and that the attributes have been provided by a genuine OAuth IdP. In such a case, the integration scheme would provide a two-level user authentication, based on selection of the correct InfoCard and user authentication at the OAuth IdP. This would offer a security advantage by comparison with the ‘native’ CardSpace personal card protocol, where the user is only authenticated once.

However, this is a significant trust assumption. We next consider two ways in which this assumption might be met.

1. The integration browser extension could be installed in a managed environment in which a user is only granted limited privileges insufficient to modify or replace the extension. However, in order for the RP to have assurance that the user platform is in such a controlled environment (and to avoid making extra changes to the RP server), the RP itself would probably need to belong to the managed environment.
2. A more widely-applicable solution would be to make use of the functionality of the Trusted Platform Module (TPM), present on a large proportion of recently manufactured PCs. Using the remote attestation mechanism, an RP could be provided with guarantees about the software state of the user platform, including the presence of the expected software implementing the CardSpace-OAuth integration scheme.

## 4.4 Client-side Integration

IdPs/RPs may not accept the burden of supporting two identity management systems simultaneously, unless there is a significant financial incentive. Currently, major Internet players do not provide any means of interoperation between identity management systems. As a result, a client-side technique for supporting interoperation could be practically useful. Supporting interoperation at the client (instead of the server(s) at the IdP and/or RP) also means that server-performance is not affected.

## 4.5 Triggering the Browser Extension

The scheme specified in section 3.2 (like the prototype implementation) uses a special sequence in a specific field of an OAuthCard to trigger the browser extension. However, other approaches could be used, e.g. the browser extension could start whenever CardSpace is triggered. In such a case, when a user submits an OAuthCard, the browser extension could offer the user the choice (e.g. via an embedded HTML form or JavaScript pup-up box) to either use CardSpace as usual or activate the integration scheme. This approach gives a greater degree of user control, and hence implements Microsoft’s first identity law [5, 13]. In addition, giving user control over whether the browser extension runs or not would enable ‘normal’ use of CardSpace. However, it is potentially a little inconvenient, since it would require users to always choose whether or not to use the integration software. Nevertheless, this effect could be mitigated if the user’s choice is stored.

## 4.6 Attribute Mapping

CardSpace and Facebook Connect use two different sets of attribute types<sup>11</sup>; this clearly causes a problem when requesting user attributes from Facebook based on a policy statement provided by a CardSpace-enabled RP. We outline two approaches to dealing with the problem.

1. We could restrict the RP to requesting only CardSpace personal card style attributes. The browser extension would then need to convert the requested attributes to Facebook style attributes, and include the converted attribute types in the request sent to Facebook. An example mapping is shown in Table 1.
2. Alternatively, the RP could be permitted to request any of the Facebook-supported attributes. If an attribute not supported by CardSpace personal cards is requested (and given that the RP permits the use of any IdP), then the browser extension would need to be configured to request it from Facebook. However if any attributes are required that are outside the set permitted in a personal card, then the CardSpace identity selector will clearly not highlight any of the personal cards.

In order to cause the selector to highlight personal cards, the browser extension must modify the RP policy. In particular, as part of step 3

---

<sup>11</sup>As stated in section 2.1.2, CardSpace personal cards only support fourteen editable attributes, whereas Facebook Connect supports many more.

Table 1: CardSpace-Facebook Connect attribute mapping

CardSpace (Personal Cards)	OAuth (Facebook Connect)
givenname	first_name
surname	last_name
emailaddress	email
dateofbirth	birthday
gender	gender
country	locale
city	location
web page	website

the browser extension must (after storing them) strip out the attributes that are outside the set supported by personal cards, and then request them from Facebook as part of step 10. Note, however, that such a modification will prevent CardSpace from operating normally in the case where a personal card is requested. Nevertheless, if the RP specifies the use of managed cards (i.e. does not permit personal cards), then CardSpace would still operate normally, since the extension will shut down if it sees such a policy statement.

Finally, we observe that in order to support the broadest range of user attributes, the browser extension could be configured to support both of the approaches described above.

## 5 Prototype Realisation

We next give details of a prototype implementation of the scheme. The description applies to Facebook Connect, an implementation of OAuth 2.0. The prototype uses Facebook Connect’s client-side flow<sup>12</sup> (i.e. the implicit ‘grant’ type).

The prototype is coded in JavaScript, chosen because its wide adoption should simplify the task of porting the prototype to a range of other browsers. It uses the Document Object Model (DOM) to inspect and manipulate HTML pages and XML documents. The JavaScript code is executed using a C#-driven browser helper object (BHO), a Dynamic-link library (DLL) module designed as a plug-in for IE. Once installed, the BHO attaches itself to IE, thus gaining access to the current page’s DOM. The

<sup>12</sup><http://developers.facebook.com/docs/authentication/>

prototype can readily be enabled or disabled using the add-on manager in the IE's Tools menu. Note that the integration plug-in does not require any changes to default IE security settings, thus avoiding potential vulnerabilities resulting from lowered browser security settings. Note also that the prototype operates with both the CardSpace and the Higgins<sup>13</sup> identity selectors without any modification.

The prototype has been successfully tested with Facebook<sup>14</sup> and an experimental implementation of a CardSpace-enabled RP.

## 5.1 Registration

Prior to use, the user must have accounts with an RP and Facebook. The user must register the RP with Facebook<sup>15</sup> (see section 3.1). The user must also create an OAuthCard, inserting the RP's identifier in the *first name* field, and the trigger word 'OAuth' in the *last name* field. Note that the Facebook URL is contained in the source code of the browser extension, and thus does not need to be included in the OAuthCard. For ease of identification, the user can give the OAuthCard a meaningful name, e.g. of the RP site. The user can also upload an image for the card, e.g. containing the logo of the RP with which it is used or simply of OAuth. When a user wishes to use the scheme with a particular RP, the user simply chooses the corresponding OAuthCard.

## 5.2 Prototype Operation

In this section we consider specific operational aspects of the prototype. We refer throughout to the numbered protocol steps given in section 3.2.

In step 3 the plug-in uses the DOM to perform the following processes.

3.1 It scans the web page in the following way<sup>16</sup>.

- (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.

---

<sup>13</sup>[http://wiki.eclipse.org/GTK\\_Selector\\_1.1-Win](http://wiki.eclipse.org/GTK_Selector_1.1-Win)

<sup>14</sup><https://www.facebook.com/>

<sup>15</sup><https://developers.facebook.com/setup/>

<sup>16</sup>Two HTML extension formats can be used to invoke the selector from a web page [10], both of which involve placing the CardSpace object tag inside an HTML form. This motivates the choice of the web page search method.

- (b) If an object tag is found, it retrieves and examines its type. If it is of type ‘application/x-informationCard’ (which indicates website support for CardSpace), it continues; otherwise it aborts.
- (c) It retrieves and stores in a cookie the name attribute of the CardSpace object tag. This is important since the RP will use this name to retrieve the token from the HTTP POST array.
- (d) It searches through the param tags (child elements of the retrieved CardSpace object tag) for the ‘issuer’ tag and examines its value; if it is ‘<http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self>’, indicating that the use of personal (self-issued) cards is acceptable, it continues<sup>17</sup>; otherwise it terminates.
- (e) It retrieves the ‘requiredClaims’ and ‘OptionalClaims’ tags from the param tags, and retrieves and stores the mandatory and optional claim types listed in these tags.
- (f) It uses the JavaScript property ‘document.location.protocol’ to discover whether HTTP or HTTPS is in use.
- (g) If necessary and if HTTP is in use, the plug-in, after keeping track of the original policy settings, modifies the RP policy so that the *first name* and *last name* claim types are specified in the ‘requiredClaims’ tag.

3.2 It adds a JavaScript function to the head section of the HTML page to intercept the RSTR (an XML-based security token).

3.3 It obtains the action attribute of the CardSpace HTML form and stores it in a cookie. This attribute specifies the URL address of a web page at the RP server to which the security 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/or *document.location.host*, are used to help reconstruct the full URL address.

3.4 It changes the current action attribute of the CardSpace HTML form to point to the newly created ‘interception’ function (see step 3.2 above).

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

---

<sup>17</sup>The plug-in also continues if the value of the ‘issuer’ tag is set to ‘any’, ‘\*’ or if the ‘issuer’ tag is absent, since the use of personal cards is acceptable in these cases.

- 8.1 It intercepts the RSTR message sent by the selector using the added function.
- 8.2 It operates slightly differently depending on whether HTTP or HTTPS is in use.
- If HTTP is used, the plug-in parses and extracts certain RSTR contents. If the *last name* field contains the word ‘OAuth’, the plug-in proceeds; if not, normal operation of CardSpace continues. It reads the *first name* field to discover the RP’s identifier.
  - If HTTPS is used, the plug-in (using a JavaScript pop-up box) asks the user whether the use of the integration protocol is required. If so, it proceeds; otherwise it terminates. On proceeding, it prompts the user to enter the RP’s identifier (as issued by Facebook). The plug-in offers the user the option to store the input values in a persistent cookie for future logins at this RP, using a plug-in-embedded checkbox.
- 8.3 It constructs an OAuth request, compatible with Facebook Connect. This involves generating a nonce and time-stamp (used to build the ‘state’ parameter), and also determining the required and optional attribute types to be requested from Facebook. The plug-in retrieves all the CardSpace-supported claim types it stored earlier (see step 3.1 (e) above). It then maps between them and the Facebook-supported attribute types, using Table 1. The mapping is done using JavaScript regular expressions, specifically the ‘match’ method with its global (g) and case-insensitive (i) parameters. The plug-in sets the value of ‘redirect\_uri’ parameter (to which Facebook will send the response) to the URL of the currently-visited RP page. In addition, it sets the value of the ‘response\_type’ parameter to ‘token’, signifying the use of the ‘implicit’ grant type.
- 8.4 It encrypts (and temporarily stores in a cookie) the RSTR and the value of the ‘state’ parameter using AES in CBC mode, using a secret key known only to the plug-in.
- 8.5 It redirects the user to Facebook along with the OAuth request, using the JavaScript inherent property ‘window.location’.

In step 10 the plug-in performs the following steps.

- 10.1 It parses the Facebook-issued response (embedded in the URL).

- 10.2 It (transparently) validates the response, including checking that the value of the ‘state’ parameter is the same as the one it generated in step 8.3, is sufficiently current, and has not been previously used. The plug-in then adds the received value to an internally stored list for use in future verifications.
- 10.3 It uses the provided access token to request and retrieve the RP-requested user attribute values from Facebook open graph<sup>18</sup> using a TLS-protected channel.

In step 11 the plug-in performs the following steps.

- 11.1 It constructs a CardSpace-like SAML security token, inserting the user attribute values received from Facebook into the token. It also embeds the signed SIIP-issued RSTR into the SAML token, after retrieving and decrypting the RSTR from the appropriate cookie (see step 8.4).
- 11.2 It creates and appends an ‘invisible’ HTML form (with the method attribute set to ‘POST’) to the current page.
- 11.3 It writes the entire SAML security token as a hidden variable into the invisible HTML form, with the name attribute of this variable set to the CardSpace object tag’s name (see step 3.1(c)). Note that the plug-in retrieves this name from the appropriate cookie.
- 11.4 It writes the end-point URL of the RP into the action attribute of the invisible form. Note that the plug-in retrieves this name from the appropriate cookie (see step 3.3).
- 11.5 Finally, it auto-submits the HTML form (transparently to the user), using the JavaScript inherent method ‘submit’.

### 5.3 Potential Issues

The plug-in must scan every HTML web page to see 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.

Some older browsers (or browsers with scripting disabled) may not be able to run the 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.

---

<sup>18</sup>[http://en.wikipedia.org/wiki/Social\\_graph#Open\\_Graph](http://en.wikipedia.org/wiki/Social_graph#Open_Graph) and <http://developers.facebook.com/docs/reference/api/user/>

## 6 Related Work

A somewhat similar scheme [1] has previously been proposed to support CardSpace-Liberty interoperation. However, unlike the scheme proposed here, the CardSpace-Liberty integration scheme does not support the exchange of identity attributes and does not operate with HTTPS-enabled websites. Two further similar schemes have recently been proposed, allowing interoperation between a CardSpace-enabled RP and a Shibboleth IdP [2] or an OpenID IdP [3].

Another scheme supporting interoperation between CardSpace and Liberty has been proposed by Jørstad et al. [12]. 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. This scheme also 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 also serves as an implicit indication of user approval for the IdP to send a security token to the RP. By contrast, the scheme proposed in this paper supports client-side interoperation between CardSpace and OAuth, does not require use of a handheld device, and does not enforce a specific authentication method.

## 7 Conclusions and Future Work

In this paper we have proposed and prototyped a means of interoperation between two leading identity management systems, namely CardSpace and OAuth. CardSpace users (indeed, users of any Information Card system) are able to obtain an assertion token from an OAuth provider, the contents of which can be processed by a CardSpace-enabled relying party. The scheme is transparent to OAuth providers and identity selectors, uses a browser extension, and requires only minor changes to a CardSpace-enabled relying party. It uses the CardSpace identity selector interface and CardSpace personal cards to enable interoperation between OAuth providers and CardSpace relying parties.

The integration scheme takes advantage of the similarity between the OAuth and 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 the OAuth owners.

Planned future work includes investigating the possibility of extending the CardSpace identity selector to simultaneously support security tokens from a variety of identity providers, such as OpenID, OAuth, Liberty, Shibboleth, as well as CardSpace remote and self-issued identity providers. Possible future work may also investigate the possibility of extending the proposed integration protocol to support CardSpace-enabled relying parties that employ security token services.

## Acknowledgements

The author is sponsored by the Diwan of Royal Court, Sultanate of Oman. The helpful comments provided by Chris Mitchell are gratefully acknowledged.

## References

- [1] Haitham S. Al-Sinani, Waleed A. Alrodhan, and Chris J. Mitchell. CardSpace-Liberty integration for CardSpace users. In Ken Klingenstein and Carl M. Ellison, editors, *Proceedings of IDtrust '10 — the 9th Symposium on Identity and Trust on the Internet, Gaithersburg, Maryland, USA, April 13–15, 2010*. ACM, New York, 12–25, 2010.
- [2] Haitham S. Al-Sinani and Chris J. Mitchell. CardSpace-Shibboleth integration for CardSpace users. In *ACNS '11 [industrial track proceedings], 9th International Conference on Applied Cryptography and Network Security, Nerja (Malaga), Spain, 7–10 June 2011*, pages 49–66, 2011. [Full version available at: <http://www.isg.rhul.ac.uk/cjm/Papers/cssifc.pdf>].
- [3] Haitham S. Al-Sinani and Chris J. Mitchell. Client-based CardSpace-OpenID interoperation. In *Proceedings of ISCIS '11 — the 26th International Symposium on Computer and Information Sciences, 26–28 September 2011, London, UK (to appear)*. To be published in the Springer Lecture Notes on Electrical Engineering (LNEE), 2011. [Full version available at: <http://www.ma.rhul.ac.uk/techreports/2011/RHUL-MA-2011-12.pdf>].

- [4] Andreas Berger. *Identity Management Systems — Introducing Yourself to the Internet*. VDM Verlag, Saarbrücken, 2008.
- [5] Vittorio Bertocci, Garrett Serack, and Caleb Baker. *Understanding Windows CardSpace: An Introduction to the Concepts and Challenges of Digital Identities*. Addison-Wesley, Reading, Massachusetts, 2008.
- [6] Eran Hammer-Lahav (editor). *The OAuth 1.0 Protocol — RFC 5849*, 2010. <http://tools.ietf.org/html/rfc5849>.
- [7] Torsten Lodderstedt (editor), Mark McGloin, and Phil Hunt. *OAuth 2.0 Threat Model and Security Considerations — draft-ietf-oauth-v2-threatmodel-00*, 2011. <http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-00>.
- [8] Eran Hammer-Lahav, David Recordon, and Dick Hardt (editors). *The OAuth 2.0 Authorization Protocol — draft-ietf-oauth-v2-20*, 2011. <http://tools.ietf.org/html/draft-ietf-oauth-v2-20>.
- [9] Jonathan Hart, Konstantinos Markantonakis, and Keith Mayes. Website credential storage and two-factor web authentication with a Java SIM. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis, and Damien Sauveron, editors, *Proceedings of WISTP '10 — Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, 4th IFIP WG 11.2 International Workshop, Passau, Germany, April 12–14, 2010*, volume 6033 of *Lecture Notes in Computer Science*, pages 229–236. Springer, Berlin, Heidelberg, 2010.
- [10] Michael B. Jones. *A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers*. Microsoft, 2008.
- [11] Michael B. Jones and Michael 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>.
- [12] Ivar Jørstad, Do Van Thuan, Tore Jønvik, and Do Van Thanh. Bridging CardSpace and Liberty Alliance with SIM authentication. In *Proceedings of ICIN '07 — the 10th International Conference on Intelligence in Next Generation Networks*. Adera, Pessac, 8–13, 2007.
- [13] Marc Mercuri. *Beginning Information Cards and CardSpace: From Novice to Professional*. Apress, New York, 2007.

- [14] Marino Miculan and Caterina Urban. Formal analysis of Facebook Connect single sign-on authentication protocol. In *SOFSEM '11 (Software Seminar): Theory and Practice of Computer Science — the 37th Conference on Current Trends in Theory and Practice of Computer Science, Slovakia, January 22–28, 2011. Proceedings of Student Research Forum*, pages 99–116, 2011.
- [15] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M. Pai, and Sanjay Singh. Formal verification of OAuth 2.0 using Alloy framework. In *Proceedings of CSNT '11 — the International Conference on Communication Systems and Network Technologies, Katra, Jammu, India, June 3–5, 2011*, pages 655–659. IEEE Computer Society, Los Alamitos, CA, 2011.

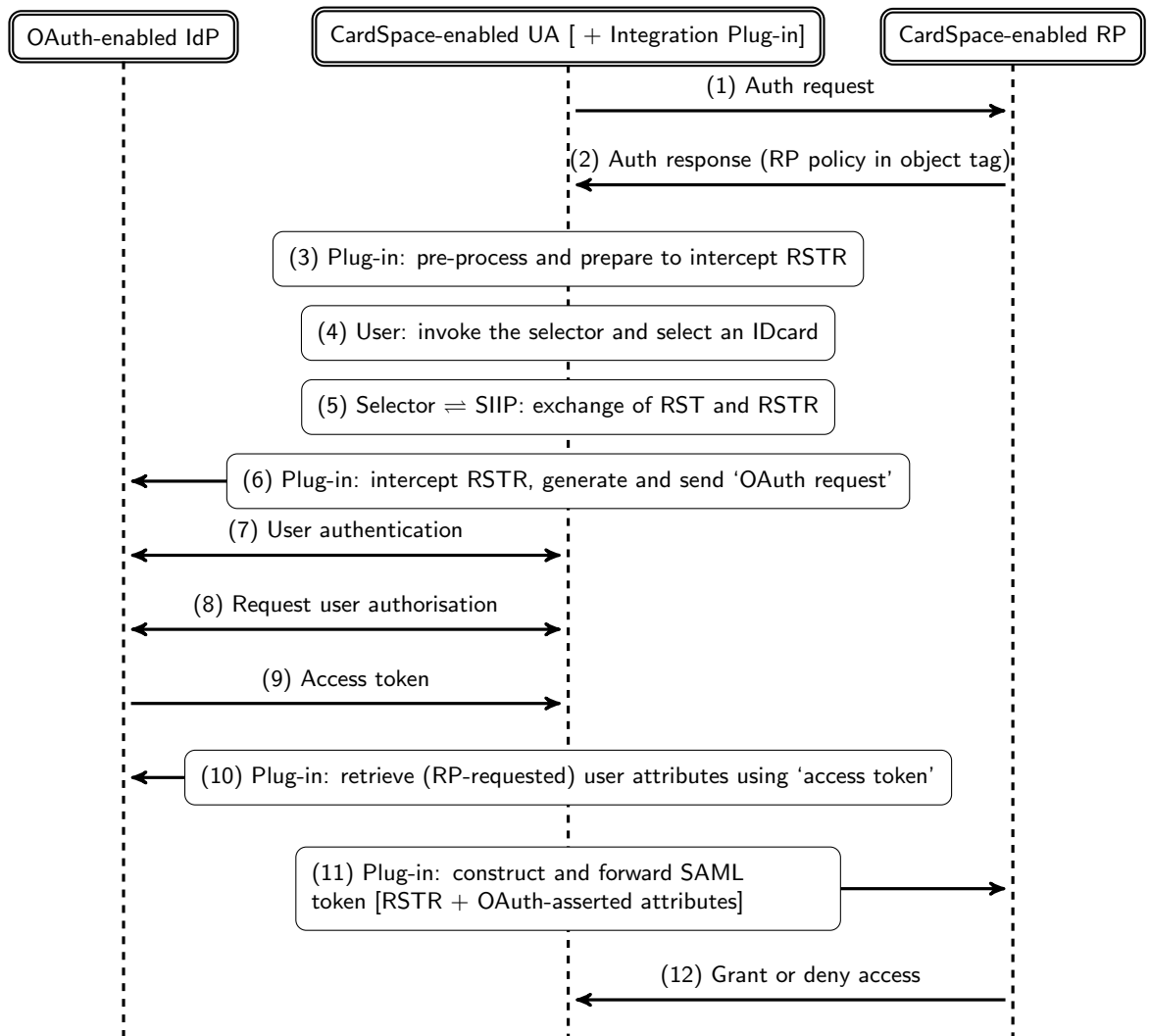


Figure 1: Simplified Protocol Exchange