New Architectures for Identity Management — Removing Barriers to Adoption*

Haitham S Al-Sinani and Chris J Mitchell
Information Security Group
Royal Holloway, University of London, UK
Haitham.Al-Sinani.2009@rhul.ac.uk me@chrismitchell.net

November 8, 2011

Abstract

In recent years a large number of identity management systems have been proposed. Unfortunately, although these systems offer the possibility of significantly improving user security, they have not been widely adopted, typically because the cost of adoption is too high for the involved parties. In this talk we consider the problem of designing identity management systems which offer security advantages but are yet easy to adopt. This involves designing combinations of security protocols and client machine software architectures that support secure identity management protocols in ways that offer simple and low cost migration paths. We describe a client-based identity management tool we call *IDSpace*, designed to be easy to adopt, and which provides a single user interface and user experience for user authentication, whilst supporting a range of existing identity management technologies. Operation of IDSpace with certain existing systems is described.

Keywords: CardSpace; identity management; identity selector; user authentication

1 Introduction

1.1 The Need for Authentication

Authentication of human users is a fundamental security requirement; indeed, it could be argued that it is *the* fundamental requirement. Despite its importance, it is almost universally acknowledged that providing user

*A very similar paper was presented at the EuroPKI '11 Workshop, held in Leuven, Belgium in September 2011, and will be published in the proceedings of that event.

authentication remains a huge practical problem. In practice, as many observers have noted (see, for example, Herley et al. [15]), we are still using passwords almost universally. Again as widely acknowledged, the use of passwords has many shortcomings, not least because users today have so many Internet relationships, all requiring authentication. In such a context, password re-use and use of weak passwords are almost inevitable.

A common approach to addressing this problem is to propose yet another new way of achieving user authentication, possibly involving a Public Key Infrastructure (PKI) [1]. However, there are already many good technological solutions. Perhaps the real problem is the insufficiently broad adoption of the solutions we already have. If so, this is partly a business and sociological issue, but perhaps it is also a problem which requires new technical thinking.

It is easy for those of us providing technological solutions to claim that this is not our problem. We provide the technology, and the business and commercial world should just get on with adopting it. However, real life is not so simple. We in the academic world should be thinking about how to devise technological solutions which are easier to adopt. As always, key issues for easy adoption are transparency, ease of use, and backward compatibility, and these factors have played a large part in the design of the system we describe here.

1.2 Identity Management

Identity (ID) management systems [8, 9, 28, 29] have been designed to simplify user authentication. An ID management system enables an Identity Provider (IdP) to support authentication of a User (and assertion of user attributes) to a Service Provider (SP). Recent years have seen the emergence of a wide range of such systems, including OpenID [24, 26], Liberty[1] [25], Shibboleth [17, 18], CardSpace [10, 22] and OAuth [16]. Each system has its own set of protocols governing communications between the main parties. As well as its own protocols, each system may also have a unique supporting infrastructure, including public key certificates, shared keys, passwords, etc. Some systems have gained a limited amount of traction recently, e.g. the use of OpenID in some sectors and Facebook's adoption of OAuth (Facebook Connect). However, the systems that have been most widely used also possess the most significant security issues (e.g. phishing vulnerabilities), and no system has broad penetration into the user community.

Many ID management systems are susceptible to phishing attacks, in which a malicious (or fake) SP redirects a user browser to a fake IdP. The user then reveals to the fake IdP secrets that are shared with a genuine IdP.

[1]The Liberty Alliance specifications have been input to the Kantara Initiative (`http://kantarainitiative.org/`)

This arises because, in the absence of a system-aware client agent, schemes rely on browser redirects.

A further problem faced by an end user is that the user experience of every ID management system is different. It is widely acknowledged that users fail to make good security decisions, even when confronted with relatively simple decisions [19]. The lack of consistency is likely to make the situation much worse, with users simply not understanding the complex privacy- and security-relevant decisions that they are being asked to make.

Finally, when using third party IdPs which provide assertions about user attributes, there is a danger that a user will damage their privacy by revealing attributes unintentionally to an SP. This is a particular threat when using systems like OAuth (e.g. as instantiated by Facebook Connect). In general, getting privacy settings right is highly non-trivial.

1.3 A New Approach

It is tempting to try to devise another new scheme which has the practical advantages of OAuth and OpenID, but yet provides robust protection against phishing and privacy loss. That is, we might wish to devise a client-based scheme with the user convenience of other systems, but which somehow avoids the fate of CardSpace[2]. However, it seems that a new solution is highly unlikely to succeed when others have failed (especially given that systems such as CardSpace have had the support of a large corporation and incorporate very attractive features). Moreover, a new system is likely to create yet another different user experience, increasing the likelihood of serious mistakes by end users. This suggests that devising yet another new system may not be the right approach.

The goal of this paper is to propose a new approach to the user authentication problem. It does not involve proposing any new protocols or infrastructures. The goal is to try to make it easier to use existing systems, and also to make their use more secure (including resistance to phishing) and privacy-enhancing (not least through the provision of a consistent user interface and an explicit user consent procedure).

The scheme we propose involves a client-based user agent. This is a single tool which supports a wide range of ID management systems yet provides a single interface to the user. The consistent user interface should maximise user understanding of what is happening and thereby reduce the risk of errors and increase user confidence. It also avoids the need for passive browser redirects, hence mitigating phishing attacks.

[2]Despite its adoption as an OASIS standard [23], in early 2011 Microsoft made a statement (`http://blogs.msdn.com/b/card/archive/2011/02/15/beyond-windows-cardspace.aspx`) implying that the project will not be pursued further.

1.4 CardSpace

One motivation for the novel scheme arises from consideration of CardSpace (and related schemes such as Higgins (`www.eclipse.org/higgins`). Before proceeding we thus need to briefly describe CardSpace.

CardSpace acts as a client-based agent, and provides a consistent card-based user interface known as the *Identity Selector*. That is, sets of user credentials (relationships with IdPs) are represented to users as cards. CardSpace also defines a set of protocols for interactions between IdPs, Clients (user machines) and SPs. The user, interacting with a browser via the identity selector, may have identities issued by one or more IdPs. Each identity is represented by an *InfoCard* held by the identity selector, and this InfoCard is the means by which the user interacts with the identity selector to choose which identity to use. Each IdP runs a Security Token Service (STS), to generate security tokens. A Self-issued Identity Provider running on the client platform is also provided to allow use of self-issued tokens.

Before issuing a token, an IdP will typically need to authenticate the user. This user authentication takes place via the local CardSpace software. There are two key advantages of such an approach: it provides a consistent user experience, and it helps to limit the possibility of phishing attacks.

The user interface of CardSpace and the underlying communications protocols are not inherently tied together. It is thus possible in principle to keep the simple/intuitive user interface, and use it as the front end for a tool which manages user credentials in a consistent way regardless of the underlying ID management system. Credential sets can then identify with which ID management system (or systems) they should be used. For example, each credential set could be stored as a self-describing XML document. Indeed, these credential sets could include username/password pairs. This series of observations provides the basis for the IDSpace scheme, which we describe next.

1.5 Organisation

The remainder of the paper is organised as follows. Section 2 presents a high-level description of IDSpace; this is followed in section 3 by a specification of its architecture. Section 4 discusses a number of key functions that an IDSpace-conformant system must provide. The detailed operation of IDSpace is given in Section 5. Section 6 describes instantiations of the IDSpace architecture when operating with specific examples of existing ID management systems. Finally, section 7 highlights possible future research directions.

2 IDSpace

We now describe IDSpace, the name of which pays homage to CardSpace. IDSpace is an architecture for a client-based ID management tool that operates in conjunction with a client web browser. A tool conforming to the architecture provides a user-intuitive and consistent means of managing a wide range of types of digital identities and credentials for user web activities. The IDSpace architecture is designed to support all existing ID management protocols, and can be used to replace existing ID management client software, including the CardSpace [10, 22], and Higgins[3] clients, Liberty-enabled client software [21], and client-based password managers.

It is important to observe that IDSpace is not an ID management system, at least in the normal sense of the term. Instead it is an architecture for a client system which enables the use of a multiplicity of ID management protocols with maximal transparency to the user (avoiding the need to install multiple ID management clients). The IDSpace architecture is designed so that conformant tools are able to work with all existing Internet SPs and IdPs without any changes to their current operation. That is, the system is transparent to all third parties.

The IDSpace architecture is designed to be platform-independent, and a prototype implementation is being developed (a partial Windows-based prototype is already operational). Implementations should be capable of being deployed on Windows, Unix, Mac, and smart phone-based platforms with minimal changes. Key parts of the system can be instantiated as browser add-ons, e.g. written in C++ and/or JavaScript, thereby maximising portability.

As with any ID management tool, the primary purpose is to enable an end user to access a protected resource. Once installed on a user device, IDSpace will execute whenever a user wishes to access a protected service using a web browser. It allows the user to select a particular ID management system from amongst those supported by the SP. It also allows the user to choose which set of credentials is to be used with this SP, where the network interactions with the SP and IdP will conform to the chosen ID management system.

An IDSpace system interacts with the user via a key component known as the *Card Selector*. This provides a visual representation of user credential sets in the form of 'virtual cards', referred to here as *credential cards* (*cCards*). The operation of this component is motivated by the CardSpace's identity selector (whose virtual cards are known as InfoCards or iCards). Higgins, which originated as an open source implementation of a CardSpace-like system, also uses the term InfoCards.

A cCard can represent any of a wide range of types of user credential,

[3] `http://www.eclipse.org/higgins/`

including:

- ready-to-use credential tokens including 'password manager' tokens containing a username/password pair, referred to as *local cCards*; and
- a pointer to a remote, credential-issuing party (an IdP), referred to as *remote cCards*.

Whilst IDSpace has a similar user interface to CardSpace and Higgins, it is also important to note certain fundamental differences. Both CardSpace and Higgins support just one set of protocols for web interactions between the user platform and third party systems. If future versions of these systems support additional protocols, then this will require corresponding modifications to SPs and/or IdPs. IDSpace, by contrast, is designed to work with almost any conceivable ID management protocol suite, and its adoption does not require any changes to third party systems (including IdPs and SPs).

IDSpace is made up of a set of self-contained components interacting with each other in a pre-defined way, thus enabling modular implementation. Such an architectural design enables new ID management protocols to be supported in a simple way by adding new software modules to an existing implementation.

3 High-level Architecture

3.1 Context of Use

As stated above, IDSpace provides a user-intuitive means for managing digital identities and credentials for user web activities, consistent across underlying ID management systems. The intended context of use is shown in Figure 1.

The parties involved, as shown in the figure, include the following.

1. The *user* interacts with a *user platform* or *hardware platform* (e.g. a PC or mobile device) in order to access services provided across the Internet. This user platform is equipped with an *operating system (OS)* on which applications execute.
2. The *IdP* provides identity services to the user. This typically involves issuing a user-specific identity token for consumption by an SP (where, although the token is intended for use by a specific user, the user's identity will not necessarily be revealed to the SP). This token will provide the SP with assurance regarding certain attributes of the user, e.g. the user identity. The IdP is located either remotely or locally on the user platform; in the latter case the IdP is referred to as a *local identity provider* (*LIP*). Examples of possible IdPs include Facebook and Google.

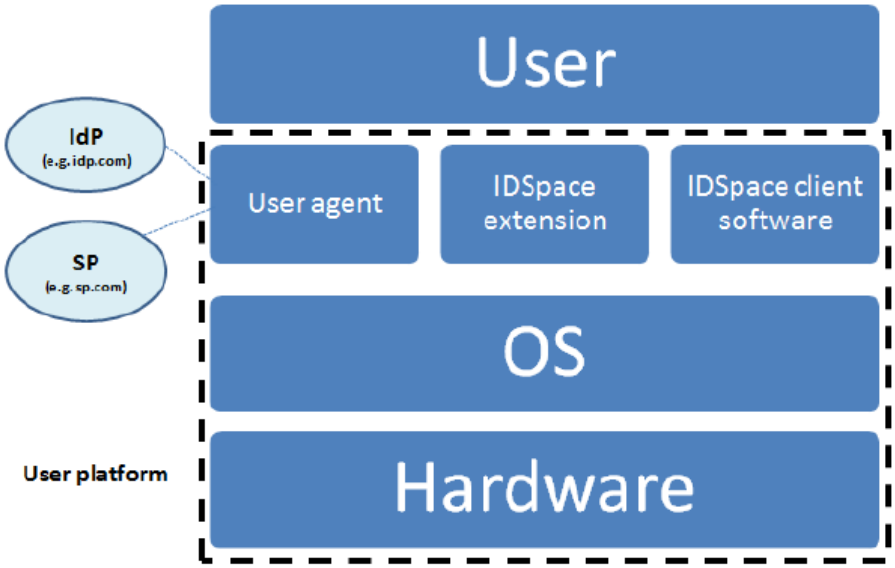

Figure 1: IDSpace Context

3. The *SP* provides services which the user wishes to access. In order to allow the user to access a protected resource, the SP will wish to be provided with verifiable statements regarding certain attributes of the user. This is typically achieved by supplying the SP with a user-specific credential or identity token issued by a local or remote IdP. (In some contexts the SP is known as a *relying party (RP)*). Examples of possible SPs include YouTube, Amazon, Facebook and Google (some parties may act as both IdPs and SPs).

4. The *user agent* (UA) is a software component employed by a user to manage interactions between the user/user platform and remote entities (IdPs and SPs). This will typically be instantiated as a web browser, such as Internet Explorer or Firefox; indeed, for the sake of simplicity, in some subsequent discussions we refer to a web browser rather than a UA. The UA processes protocol messages on behalf of the user, and prompts the user to make decisions, provide secrets, etc.

5. The *IDSpace client software*, implementing part of the IDSpace architecture, interacts with the user via a graphical user interface (GUI). This GUI allows the user to select a particular credential set (represented as a cCard) for use in a specific transaction with an SP. The application also interacts with a web browser, and, where necessary, with remote entities.

6. The *IDSpace extension* (or the *IDSpace browser extension*), implementing part of the IDSpace architecture, supplements the functionality of the UA. It is made up of a set of modules performing specific tasks, e.g. scanning a webpage for a username-password login form. The IDSpace extension exchanges data with the client software via the browser, and, where necessary, interacts with the user.

3.2 IDSpace Components

Figure 2 shows the relationships between the main components of IDSpace, including the primary information flows. The dotted line shows the limits of the browser extension. Note that, although shown as part of the browser extension, the *Activator* could also be implemented as an independent component. This is because, in certain ID management systems e.g. CardSpace, the SP webpage must implement certain X/HTML tags to enable this component to perform its task (see below). However, it is also possible for a browser extension to add such tags.

The remaining components, apart from the 'web browser' and 'remote IdP', represent the IDSpace *client software*. Note that the boxes marked 'Other ...' refer to other IDSpace components, which, although covered in the text, are not shown in the figure.

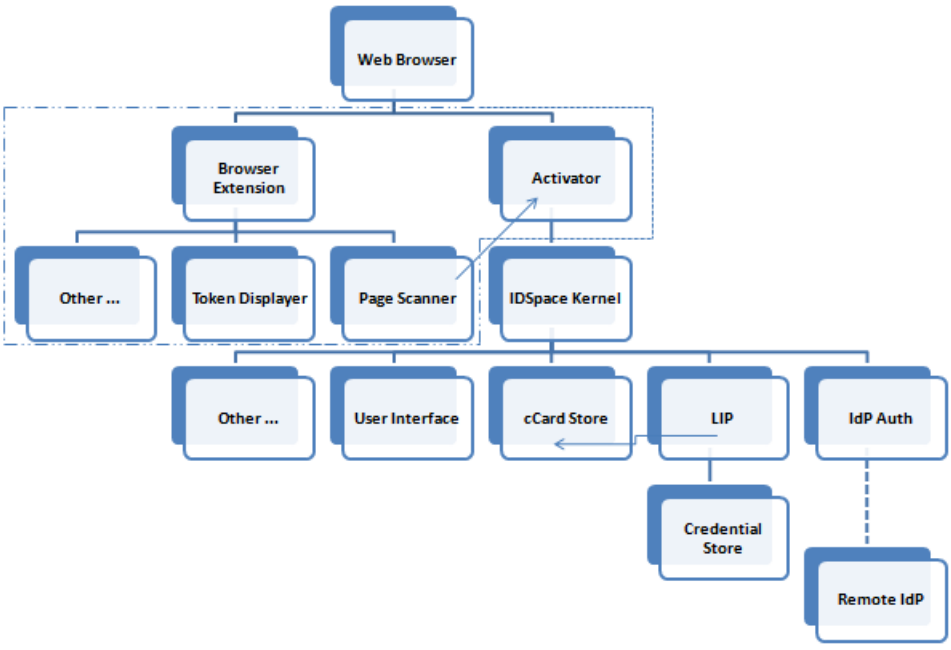

Figure 2: IDSpace Components

The two primary elements of the IDSpace architecture, i.e. the IDSpace

client software and the IDSpace extension (as introduced in section 3.1), are now discussed in greater detail.

3.2.1 Client Software

The client software, a stand-alone application, is made up of the following components.

cCards A cCard is a (relatively non-sensitive) XML document corresponding to a set of user credentials (or, more generally, to a set of user private information). A cCard indicates the types of personal information in the set, and also the type (or types) of ID management system with which the cCard can be used. However, it does not contain the personal information itself. cCards can be *local*, in which case they are generated by the LIP, or *remote*, in which case they are generated by a remote IdP.

cCard Store This is a protected local store for *cCards*. The nature of the protection provided for stored cCards will depend on the implementation environment. For example, protection could involve the use of cryptography, physical protection and/or logical protection (as provided by the OS).

Credential Store This is a protected local store for sensitive data, such as personal information, certificates, user passwords, etc., associated with local cCards. It is used by the LIP. Note that, in practice, the *Credential Store* and the *cCard Store* could be combined. As is the case for the cCard store, the nature of the protection provided will be implementation-dependent, and could involve the use of cryptography, physical protection and/or logical protection.

Settings Store This is a local store for (relatively) non-sensitive data such as system state, system/user settings, user preferences, etc.

IDSpace Kernel This is the central component of IDSpace. It runs locally on the user platform, handling communications with and between other components of IDSpace. In particular, it performs the following functions.

- It receives and processes the security policy provided by the *Activator*.
- It retrieves the cCards from the *cCard Store*, and checks which of them meet the requirements of the SP's security policy.
- It invokes the IDSpace *User Interface* in a private desktop window, and displays the cCards that meet the SP's policy requirements.

- If a remote cCard is chosen, it retrieves the security policy of the relevant remote IdP by initiating a connection with it.
- It communicates with the user-selected IdP (either a remote IdP or the LIP) to obtain an identity token, where necessary using the *IdP Auth* component.

User Interface This component, which incorporates the **IDSpace Card Selector**, is the main means by which an end user interacts with the IDSpace client software. Its tasks include the following.

- It displays the identity of the SP website to the user, and indicates whether the website has been visited previously. If the website is being visited for the first time then it allows the user to either continue or terminate.
- It displays the available cCards (it might display all the cards and highlight those that meet the SP site policy, or it might only display those meeting the policy). Note that the cCards are displayed in the *Card Selector*.
- It allows the user to review the contents of a cCard.
- It allows the user to generate and modify 'local' cCards (as opposed to 'remote' cCards generated by remote IdPs) — in doing so it provides an interface to some of the functions of the LIP.
- It allows the user to import a cCard provided by a remote IdP.
- It asks a user for explicit consent before providing potentially sensitive information to an SP.
- It allows the user to set preferences for future operation of the system. These preferences are stored in the *Settings Store*.

LIP This provides the functionality of an IdP, but is resident on the user platform. Like any IdP, the LIP can generate identity tokens. These tokens can be retrieved by the *IDSpace Kernel*. The LIP stores user-attribute values and other sensitive user data in the *Credential Store*.

IdP Auth This authenticates the user to a remote IdP, if a remote cCard is selected. It uses the *User Interface* to prompt the user to enter the required credentials, e.g. username and password, and then submits them to the IdP. By doing so it enables a consistent and simple user authentication interface to be provided to the user, even when a range of different identity protocols are being used. It also supports IdP-specific protocol interactions, e.g. to create requests for specific types of token.

Networker This initiates a direct online connection between the client software and a remote server (i.e. not involving the browser).

3.2.2 Browser Extension

The IDSpace extension, typically implemented as a browser add-on, includes the following modules.

Page Scanner This browser extension module scans the SP website login page in order to discover the identity system(s) it supports. It passes the results of the scan to the *Identity System Selector*.

Activator This is a (logical) bridge between the client browser and the *IDSpace Kernel*. Its tasks include the following.

- It informs the user that the IDSpace system can be used.
- It enables the user to activate the Card Selector.

Identity System Selector This browser extension module enables the user to select the identity management system to be used from amongst those supported by the SP website. The precise operation of this component will depend on the implementation of the IDSpace architecture.

If more than one identity system is available, the Identity System Selector could ask the user to either choose an identity system immediately or defer the selection until the point at which a cCard is selected (using the *IDSpace Card Selector*). It might also provide a means to store the user answer (in the *Settings Store*) for future authentication attempts.

It passes the user response to the *Data Transporter*.

Data Transporter This browser extension module provides the means to exchange data between components of the IDSpace architecture, including the following.

- It is responsible for the transfer of metadata regarding the SP page (e.g. the discovered and selected identity system(s), the identity of the SP, the SP website policy requirements, etc.), to the *IDSpace Kernel*. For example, if the user indicates that IDSpace is to be used, it passes the security policy of the SP website to the IDSpace Kernel.
- It transfers data from the IDSpace Kernel to the browser. For example, if IDSpace obtains or generates an identity token during the authentication process, it gives the token to the browser which dispatches it to the SP website.

Token Displayer This browser extension module displays an indication of the contents of an IdP-generated identity token to the user. This helps the user to decide whether or not to allow the token to be passed to the SP. This function can only be provided if the token is not:

- encrypted in such a way that only the SP can read it (e.g. using an SP's public key); and
- transmitted via a (direct) IdP-SP channel, i.e. the token must pass via the client platform.

4 Supporting Functionality

We next discuss a number of key functions that an IDSpace-conformant system must provide. For many of these functions we outline multiple approaches to implementation.

4.1 Identity System Discovery

IDSpace must be able to determine which ID management systems are supported by an SP website. This can be accomplished in a number of different ways, including the following.

1. IDSpace could scan the visited page for HTML/XHTML tags that are associated with specific ID management systems. For example, the string:

   - 'application/x-informationCard' indicates support for CardSpace; and
   - 'openid_url' and/or 'openid_identifier' indicates support for OpenID.

   The benefits of such an approach include complete transparency, albeit at the cost of performance (because IDSpace must scan every web page).

2. IDSpace could ask the user which ID management systems the page currently supports. The benefits of such an approach include accuracy and higher performance, at the cost of transparency and user convenience (although the user's choice could be stored in the *Settings Store* for future logins).

3. IDSpace could employ a hybrid approach based on a combination of the above two options, e.g. so that if the first option fails then it resorts to the second option.

4.2 Identity System Selection

Having learnt which ID management system(s) an SP supports, IDSpace must allow the user to select which system to use for the current transaction. Such a process could take place before or after invocation of the *IDSpace Card Selector*. We next consider these options in greater detail.

1. **Prior to selector invocation**. IDSpace could allow the user to choose the ID management system in one of the following ways.

   - IDSpace could embed a descriptive icon (logo, image, link or button) in the web page for each available system, and require the user to select one (e.g. by clicking the selected icon). Whilst this approach is intuitive and transparent, it could damage the appearance of the page, particularly if there are many logos to embed.
   - IDSpace could ask the user which system they wish to use by embedding forms in the page or by triggering pop-up boxes. The benefits of such an approach would include accuracy and higher performance, at the cost of minor user convenience.
   - IDSpace could add an ID management system selection option to the in-page context menu (i.e. the menu that appears as a result of right-clicking). Once such an option is selected, a list of ID management systems would be displayed, allowing the user to select one. Whilst this might be transparent, it might not be so intuitive to end users.
   - IDSpace could extend the browser frame[4], e.g. by adding a browser icon, bar or menu. Once the added icon (or bar or menu) has been selected, the user could choose one of the systems currently supported by the SP. Whilst this may be transparent, modifying the browser frame could be somewhat intrusive to the end user.

2. **After selector invocation**. The IDSpace Card Selector could display the currently supported ID management systems, allowing the user to select one. This choice could be combined with a display of the available cCards (if any) associated with each of the systems. In the latter case, the selector window could be partitioned so that each section displays an ID management system along with a previously used cCard for that system; a clickable option could be used to request the display of other available cCards. This approach would be transparent, convenient and would avoid making changes to web browsers or web pages. However, it would require more processing, and hence could adversely affect client platform performance.

[4]Both the browser frame and the browser-displayed web page could be extended. Browser extensions could, for example, create lightweight buttons, menu extensions, and in-process browser helper objects. The browser frame could be extended using band objects, and the web page content could be enhanced with, for example, ActiveX Controls or similar technologies [13].

4.3 Card Selector Invocation

In response to a user action, IDSpace must be able to invoke the IDSpace card selector. This involves embedding IDSpace support in the SP web page using a browser extension (see above).

4.4 IdP Discovery

IDSpace must help the user discover an IdP from which the user can obtain a suitable identity token. This process varies considerably depending on the ID management system in use. Specific approaches must therefore be devised for each supported system. The primary goal of the architecture is to allow this to take place in a way that is both as user-transparent as possible and gives a view of the process to the user that is consistent across ID management systems.

4.5 cCard Storage

The format of cCards must be sufficiently flexible and self-contained in order to allow cCard storage in a variety of locations, and to support portability. We assume that cCards will be protected while stored (where, as stated previously, the nature of this protection will be implementation-dependent).

cCards could be stored on various media, including:

- local file systems, which would give good performance and allow fast retrieval;
- remote web servers ('the cloud'), which would give a roaming capability;
- portable user devices such as mobile phones or smart cards, which would also provide a roaming capability.

4.6 cCard Format

Each cCard will contain an identifier indicating the ID management system with which it can be used (in principle a cCard could have many such identifiers). We suppose here that cCards are encoded using XML (as is the case for CardSpace InfoCards). A single XML schema could be devised encompassing all supported ID management systems. This would have the advantage that the identity system identifier (discussed immediately above) could form part of the encoding of a cCard. Other methods of encoding could also be used, such as JSON (`http://www.json.org/`).

4.7 cCard Content

The content of a cCard will vary depending on the ID management system with which it is to be used. However, the types of content listed below are likely to be contained in almost all cCards.

1. **A list of supported attribute types**, e.g. age, password, first name, last name, the values of which are known by the IdP, and for which the IdP will be prepared to generate an identity token. The actual claim values are not stored by the Card Selector; they are either stored by the remote IdP or by the LIP. The LIP will store the values in the protected Credential Store. Protection could, for example, involve implementing the Credential Store on a separate device such as a smart card, or using a Trusted Platform Module (TPM) [14] to provide encrypted storage.

2. **A list of supported token type(s)**, indicating which type(s) of identity token (e.g. SAML, username-password) the IdP(s) associated with the card are capable of issuing.

3. **IdP location**, including the URI/URL address(es) of the (remote or local) IdP(s).

4. **IdP authentication method(s)**, specifying the method(s) employed by the IdP to authenticate the user.

5. **Display information**, e.g. an image and or a name for the cCard.

4.8 Process Isolation

Where possible, the IDSpace processes should be isolated from other processes to maximise the security and privacy of data handled by IDSpace. For example, on a Windows platform the IDSpace Card Selector could be invoked in a private desktop session.

4.9 Authentication Methods

The IDSpace architecture allows the user to be authenticated to an IdP using a wide range of different authentication methods. The ease with which additional methods can be supported depends on precisely how user authentication to a remote IdP is supported by IDSpace. We consider three main possibilities.

1. IDSpace could control all communications between the user and the remote IdP. That is, all requests for authenticating information by the IdP could be made to the user by IDSpace (specifically by the *IdP Auth* component, as described in section 2), and the supplied information

could then be forwarded by IDSpace to the remote IdP. Adding a new authentication method would require adding functionality to the implementation of IDSpace executing on the user platform. This is the approach adopted by CardSpace, currently deployed versions of which support four authentication methods.

New user authentication techniques could be added in a modular fashion, as and when they are required. Whilst this would clearly add to the cost of deploying and maintaining an IDSpace implementation, for a widely deployed system this does not seem such an unreasonable approach (given that the number of authentication methods seems unlikely to grow very rapidly). Such an approach would have the advantage of user transparency and would enable the provision of a consistent user interface for the authentication process, and is hence the preferred option.

2. IDSpace could cause the task of user authentication to be performed at the IdP rather than via the IDSpace User Interface (i.e. using the IdP Auth component). That is, whenever a remote IdP requires user authentication (e.g. prior to issuing an identity token), IDSpace would redirect the UA (web browser) to the IdP, allowing the IdP to directly authenticate the user using a method of the IdP's choice. Although such a simple approach would minimise the maintenance cost for IDSpace, the user would lose the consistent experience provided by the IDSpace User Interface.

3. IDSpace could employ a hybrid approach. The default would be the first approach outlined above. IDSpace could support a set of widely-adopted (possibly standardised) authentication methods; new methods could be added as and when it is deemed appropriate. However, if an IdP wishes to use a technique not supported by IDSpace, then IDSpace could redirect the UA (web browser) to the IdP for 'direct' authentication.

5 IDSpace Operation

5.1 Initialisation

Prior to use of IDSpace, the following preparatory steps must be performed.

- The IDSpace components, including the browser extension and the client software, must be installed on the user platform.

- The user must install cCards in the cCard Store on the user platform. As noted above, these cCards can be created by either a local or a remote IdP. We briefly consider the two cases.

  - Local cCards are created using the LIP. Once it has created a cCard, the LIP will insert it in the cCard Store, and the corresponding user data will be added to the Credential Store. A user could also choose to create a local cCard during use of IDSpace.
  - Remote cCards are created by remote IdPs. Typically the creation of such a cCard will occur via an 'out of band' process, i.e. a process completely independent of the operation of IDSpace, perhaps involving the user completing a registration process using the IdP website. The resulting cCard will be provided to the user, and the user can then arrange for it to be imported into IDSpace using the IDSpace User Interface.

- For ease of identification, the user can personalise a cCard, e.g. by giving the card a meaningful name, and/or uploading an image representing the card to be displayed by the User Interface.

5.2 Protocol Flows

We now describe the operation of IDSpace. It is important to note that some parts of the operation of IDSpace will vary depending on the specific ID management system in use. The operation of IDSpace in the case of two widely discussed ID management systems is described in the next section.

1. UA → SP: HTTP/S GET Request. A user employs the UA to navigate to an SP login page.

2. SP → UA: HTTP/S Response. A login page is returned to the UA.

3. IDSpace Browser Extension→ UA: Page Processing. Certain IDSpace browser extension modules (as described below) perform the following processes on the login page provided by the SP.

   (a) Page Scanner → UA: Page Scanning. The Page Scanner module scans the login page to discover which ID management system(s) are supported by the SP (from amongst those supported by IDSpace). It passes the identifiers of the supported systems to the Identity System Selector. If no ID management system is identified, the Page Scanner could embed an icon in the browser frame to allow the user to inform IDSpace if there is an SP-supported ID management system available that has been missed.

   (b) Identity System Selector → UA. The Identity System Selector module uses the results passed to it by the Page Scanner. If more than one ID management system is discovered, then (depending on the implementation) the Selector could ask the user to select one. Alternatively, the decision could be deferred and made using

the *IDSpace Card Selector*. The advantages and disadvantages of the two approaches are discussed in section 4.2. A further alternative approach would involve the user deciding at which stage to make a choice.

The module might also offer to store any choices made by the user (in the *Settings Store*) for managing future authentication attempts. The module finally reports all the results to the *Data Transporter* module (see below).

(c) Activator $\rightleftharpoons$ UA: Card Selector Activation. The *Activator* module provides a means for the user to activate the IDSpace Card Selector. How this is achieved is implementation specific (options are discussed in sections 4.2 and 4.3). This involves embedding IDSpace-enabling tags and an IDSpace security policy in the login page. The embedded policy is subsequently used by the IDSpace User Interface to help it decide which cCards should be displayed for possible use.

4. User $\rightarrow$ UA: Card Selector Invocation. The user performs an action which invokes the IDSpace Card Selector. The precise way in which this occurs is implementation specific (options are discussed in section 4.2).

5. Data Transporter $\rightarrow$ IDSpace Kernel: Passing Metadata. The Data Transporter module passes the necessary metadata (e.g. the identified and/or selected identity system(s), the SP identity, the SP policy requirements, etc.) to the IDSpace Kernel.

6. IDSpace Kernel $\rightleftharpoons$ Card Selector: SP Identity. The IDSpace Kernel examines the SP identity (as received from the Data Transporter module in the previous step), including noting whether or not the SP uses HTTPS and whether or not the user has visited this particular SP before. The IDSpace Kernel uses the IDSpace Card Selector to:

   (a) identify the SP to the user; and

   (b) ask the user whether to continue or terminate the protocol.

   Depending on the user answer, IDSpace either continues or terminates the protocol. To assist in user decision-making, the Card Selector could indicate key security-relevant features of the SP to the user, e.g. using visual cues. In particular, it could indicate whether or not the SP:

   - uses HTTPS;
   - possesses an extended evaluation certificate;

- has been visited before; and/or
- requires a large number of, or particularly sensitive, user attributes.

The Card Selector could also offer the user a recommendation as to whether or not to continue, based on user policy settings and the SP's security properties.

7. IDSpace Kernel ⇌ IDSpace Components. The IDSpace Kernel evaluates the received metadata in order to learn which actions to take. If the user has already chosen an ID management system, then the following processes take place.

   (a) IDSpace Kernel ⇌ cCard Store: cCards Retrieval. The IDSpace Kernel retrieves the appropriate cCards (possibly none) by comparing the received metadata with the available cards. Note that the retrieved cards are specific to the user-selected ID management system.

   (b) IDSpace Kernel → Selector: Displaying cCards. The IDSpace Kernel passes the retrieved cCards to the Card Selector so that they can be displayed to the user. cCards previously used with this SP (if any) could be displayed more prominently than the others.

   If the user has not yet chosen an ID management system, then the following processes take place.

   (a) IDSpace Kernel ⇌ cCard Store: cCard Retrieval. The IDSpace Kernel retrieves the appropriate cCard(s) by comparing the received metadata with the available cards. Note that cards will be retrieved for all the SP-supported ID management systems.

   (b) IDSpace Kernel → Card Selector: Displaying SP-supported ID Management Systems + cCards. The Kernel passes the SP-supported ID management systems, along with the matching cCards (if any), to the Card Selector to be displayed to the user. The Card Selector displays the list of supported ID management systems, together with the available cCards, indicating which cards have been used previously with this SP (it could also indicate which ID management systems have been previously used with this SP).

   Depending on the implementation and the number of systems and cards to be displayed, the Card Selector might only display the cards previously used. In such a case it would need to indicate that other cards are also available, and would need to provide a means to retrieve them.

In both cases, the Card Selector should also allow the user to create a new local cCard (if the relevant ID management system supports such cards).

8. User → Card Selector: Selecting/Creating cCards. The user selects (or creates) a cCard.

9. Card Selector → IDSpace Kernel: User Action Results. The Card Selector reports the results of the user actions back to the IDSpace Kernel.

10. IDSpace Kernel ⇌ IDSpace Components. The IDSpace Kernel evaluates the results received from the Card Selector, and takes the appropriate steps.

    If the user has chosen to select an existing cCard, then the following processes take place.

    (a) The IDSpace Kernel determines whether an IdP (local or remote) needs to be contacted. If not, control is passed to step 13. If so, the protocol continues.

    (b) The IDSpace Kernel determines the IdP (local or remote) that must be contacted in order to enable the user to obtain the identity token required by the SP. This also includes determining the nature of the information regarding the user (e.g. login credentials) that must be supplied to this IdP.

    (c) IDSpace Kernel ⇌ Card Selector: Display IdP Identity. If this IdP has not previously been used, or if it does not use HTTPS, the IDSpace Kernel uses the Card Selector to obtain user consent before sending the IdP any information. This step is designed to mitigate the risks of phishing attacks. In such a case the Card Selector reports the user response back to the Kernel.

    (d) If user consent has been obtained, the Kernel now passes a token request to the IdP. The token request may have been received from the SP, or, if necessary, the IDSpace Kernel creates the request.

    If the user has chosen to create a local cCard, the following processes take place.

    (a) IDSpace Kernel ⇌ Selector GUI. The Kernel invokes a special Card Selector window to allow the user to enter the necessary data. This would typically include allowing the user to personalise the cCard, e.g. uploading a card image, entering a card name, etc. Such steps would enable the card to be readily recognisable.

(b) IDSpace Kernel ⇌ Card Creation Module (in the LIP): Card Creation. The Kernel instructs the Card Creation module to create an XML-based cCard using the user-inserted data. The Card Creation module returns the newly-created card to the Kernel.

(c) IDSpace Kernel ⇌ cCard Storage Module: Card Storage. The Kernel sends the cCard to the Card Storage module for permanent storage; the Card Storage module reports back to the Kernel whether or not the operation has been successful.

(d) IDSpace Kernel ⇌ Card Selector. The Kernel treats the newly-created cCard as a user-selected cCard and step 10a repeats.

11. IDSpace Kernel ⇌ IdP. One of the following processes takes place, depending on whether the selected IdP is local or remote.

    - If a remote IdP is selected, and if such information is required by the IdP (and is not already stored by IDSpace) then the IDSpace Kernel prompts the user to enter the relevant IdP credentials using a special credential screen. If this fails, e.g. if the Kernel does not support the IdP authentication method, or if the user-selected ID management system dictates that the UA must be redirected to the IdP, then the Kernel redirects the UA (web browser) to the remote IdP along with an authentication request. In the latter case the IdP can authenticate the user directly using an authentication method of its choice.

      If user authentication is successful, the IdP issues an identity token.

    - If a local IdP is selected, then the Kernel constructs a token request and sends it to the LIP. The LIP responds with an appropriate identity token.

12. Token Displayer Module ⇌ User. If an ID management system other than CardSpace is in use, then the Token Displayer module intercepts, analyses, and displays information about the identity token before releasing it to the SP, and seeks user consent for release. If consent is denied, then the protocol is terminated. Note that this assumes that the token is not end-to-end encrypted to the SP and that it is not sent via a direct IdP-SP channel.

    If CardSpace is in use, then the CardSpace IdP will send back a display token along with the real token, which the Kernel can instruct the Card Selector to display to the user, prior to obtaining user consent.

13. IDSpace Kernel → UA → SP: Passing Identity Token. The identity token is passed to the UA, which forwards it to the SP.

14. SP → User: Grant/Deny Access. The SP validates the token, and, if satisfied, grants access to the user.

6 Mapping Specific Protocol Architectures onto IDSpace

ID management systems can be classified according to how the SP communicates via the client with the IdP. There are two main ways in which this can be achieved, namely by using an HTTP redirect or involving an active client.

1. **Redirect-based Systems**. In such a scheme, the UA is redirected by an SP to an IdP (and vice versa). In such a case the UA is essentially passive, and does not need to be aware of the ID management system in use. One major disadvantage is that a malicious SP can redirect the UA to a malicious IdP impersonating an expected IdP (e.g. to fraudulently obtain user credentials). Example systems of this type include OpenID, Liberty (browser-post profile), Shibboleth, and Facebook Connect.

2. **Active Client-based Systems**. In schemes of this type, the UA must incorporate an 'active client', which acts as an intermediary between SPs and IdPs, and which must be aware of the ID management system in use. Typically all communications between an SP and an IdP occur via this active client, and there is no need for direct SP-IdP communication. Depending on the details of the system in use, the active client can prompt the user to select a digital identity, choose an IdP, review (and perhaps modify) an identity token created by the IdP, and approve a transaction. Phishing attacks are mitigated since an SP cannot redirect the UA to an IdP of its choosing. The active client can also provide a consistent user experience, and its existence helps to give the user a greater degree of control. Examples include CardSpace and Liberty (when using a Liberty-enabled client (LEC)).

We now describe how two specific examples of ID management systems can be mapped onto the IDSpace architecture. We consider OpenID [24, 26] and Liberty (using a LEC) [20] since they are widely discussed examples of the above two models. We also briefly look at CardSpace support. These descriptions are intended as examples; this is not the only way in which the systems concerned could be supported using IDSpace.

6.1 IDSpace and OpenID

6.1.1 cCards

Either prior to, or during, use of IDSpace, the user must create an OpenID-specific cCard. This cCard must contain one required field, and may also contain one optional field, as follows.

1. The single **required field** must contain the user's OpenID.
2. The **optional field** contains the identifier of the user's OpenID IdP.

The cCard contains a unique, OpenID-specific identifier, and is stored in the secure cCard store, possibly in an OpenID-specific location (e.g. to allow faster look-up/retrieval).

6.1.2 Protocol

We now describe one way in which IDSpace could support OpenID. Steps 3b, 4–9, 10a–d (second series), 13 and 14 of the IDSpace-OpenID-specific protocol are the same as steps 3b, 4–9, 10a–d (second series), 13 and 14, respectively, of the generic IDSpace protocol given in section 5.2, and hence are not described here. Whenever prompted to select/create/import a cCard, it is assumed that the user will select/create/import an OpenID-specific cCard.

1. UA → SP: HTTP/S GET Request. A user navigates to an OpenID-enabled SP.
2. SP → UA: HTTP/S Response. A login page is returned containing an OpenID form.
3. IDSpace Browser Extension→ UA: Page Processing. The browser extension performs the following processes on the login page provided by the SP.
   (a) Page Scanner Module → UA: Page Scanning. The Page Scanner module searches the login page for an OpenID login form; such a form can be identified by searching for an input field named 'openid_url' and/or 'openid_identifier'. (The Page Scanner module also scans the page for triggers for any other ID management systems supported by IDSpace.) Finally, the module passes the search results to the Identity System Selection module.
   (c) Activator ⇌ UA: Selector Activation. The *Activator* module performs the following processes.

i. It embeds IDSpace-enabling tags in the SP-provided login page, including a security policy statement in the format required by IDSpace. This policy statement must request OpenID-specific cCards.

ii. It adds a special function to the SP-provided login page to intercept the identity token that will later be returned by the IDSpace Kernel.

iii. It employs certain (implementation-dependent) means to enable the user to activate the IDSpace Card Selector (see sections 4.2 and 4.3); e.g. it might cause a special icon to appear above the submit button with the property that clicking this icon invokes the selector.

10. IDSpace Kernel $\rightleftharpoons$ IDSpace Components. The IDSpace Kernel evaluates the results (as provided by the Card Selector) in order to take appropriate actions. If the user has chosen to select an existing OpenID-specific cCard, then the following steps are performed.

    (a) The IDSpace Kernel retrieves the cCard and passes it to the UA.

    (b) The Browser Extension parses the received cCard, retrieving the value of the user's OpenID and (if present) the OpenID IdP.

    (c) The Browser Extension temporarily stores the OpenID IdP value.

    (d) The Browser Extension adds the user's OpenID identifier to the OpenID form, and submits the form back to the SP.

    (e) The SP performs an IdP discovery process. Once the OpenID IdP has been discovered, the SP generates an OpenID authentication request and attempts to redirect the user's browser to the IdP.

    (f) The Browser Extension intercepts the SP-initiated OpenID authentication request, and compares the value of the OpenID IdP in this request with the OpenID IdP value it stored in step 10c. If they match, the process continues (with redirection of the UA to the IdP). If not, the Browser Extension could either terminate or warn the user of a possible phishing threat and ask whether or not to continue.

    (g) From this point on, OpenID operates as it would do in the absence of IDSpace, except for the final check in step 12 (see also the discussion below). In particular the user experience is OpenID-specific, and the user will see the OpenID IdP's authentication page.

11. OpenID IdP $\rightleftharpoons$ User. If necessary (authentication may be unnecessary if an IdP-user session already exists), the OpenID IdP authenticates

the user. If successful, the OpenID IdP requests permission from the user to send the OpenID assertion token to the SP.

12. Token Displayer $\rightleftharpoons$ User. When the OpenID IdP attempts to redirect the UA back to the SP, the Token Displayer module intercepts, analyses, and displays the OpenID identity token to the user before releasing it to the SP. If user consent is obtained, then the protocol continues; otherwise it terminates. Note that this is possible since the OpenID token provided by the IdP is not encrypted.

The above example describes only a partial integration of OpenID with IDSpace. We believe it is possible to replace direct authentication of the user by the OpenID IdP with a process mediated by IDSpace (specifically using the *IdP Auth* module). This would enhance the user experience by making the user authentication process consistent across different ID management systems. However, whilst the system described above has been successfully prototyped, the latter enhancement has not been implemented, and hence its practicality remains untested.

6.2 IDSpace and Liberty (LEC)

6.2.1 LECcards

Either prior to, or during, use of IDSpace, the user must create a Liberty-specific cCard. This cCard must contain one required field, and may also contain one or more optional fields, as follows.

1. The single **required field** must contain the identifier of the user's Liberty IdP.

2. The **optional field(s)**, could contain other alternative 'backup' Liberty IdPs.

The cCard contains a unique, LEC-specific identifier, and is stored in the secure cCard store, possibly in a Liberty (LEC)-specific location (e.g. to allow faster look-up/retrieval).

6.2.2 IdP Auth Functionality

The *IdP Auth* module is part of the client software. When supporting Liberty (LEC profile) its functionality includes the ability to handle token requests in Liberty format (received from Liberty SPs and sent to Liberty IdPs) and also the means to parse and process token messages received from a Liberty IdP. It makes use of the *Networker* module to communicate with the IdP and SP.

6.2.3 Protocol

We now describe one way in which IDSpace could act as a Liberty client.

Steps 3(b,c), 4–9, 10a–d (second series), 13 and 14 of the IDSpace-LEC-specific protocol are the same as steps 3(b,c), 4–9, 10a–d (second series), 13 and 14, respectively, of the generic IDSpace protocol given in section 5.2, and hence are not described here. Whenever prompted to select/create/import a cCard, it is assumed that the user will select/create/import a Liberty-specific cCard.

1. UA → SP: HTTP/S GET Request. A user navigates to a LEC-enabled SP.

2. SP → UA: HTTP/S Response. A login page is returned containing an option (e.g. a button, link, or image) to use Liberty (we use Liberty here and below to mean Liberty using the LEC profile).

3. IDSpace Browser Extension→ UA: Page Processing. The Browser Extension performs the following processes on the login page provided by the SP.

   (a) Page Scanner Module → UA: Page Scanning. The Page Scanner module searches the login page for a distinguishing feature that indicates support for Liberty. (The Page Scanner module also scans the page for triggers for any other ID management systems currently supported by IDSpace.) Finally, the module passes the search results to the Identity System Selection module.

10. IDSpace Kernel ⇌ IDSpace Components. The IDSpace Kernel evaluates the search results (as provided by the Card Selector) in order to take appropriate actions. If the user has chosen to select an existing Liberty-specific cCard, then the following steps are performed.

    (a) The IDSpace Kernel retrieves the cCard, and passes it to the *IdP Auth* module.

    (b) The IdP Auth module parses the received cCard, retrieving the values of the LEC IdP(s) and temporarily stores them.

    (c) IDSpace IdP Auth → SP: HTTP Request. The IdP Auth module issues an HTTP request to the SP containing a Liberty-enabled header (or with a Liberty-enabled entry in the *User-Agent* header).

    (d) SP → IdP Auth: HTTP Response + Authentication Request. The SP generates a Liberty authentication request and sends it to the IdP Auth module in the body of the HTTP response. The SP could choose to include a list of IdPs it knows about in the request.

(e) The IdP Auth compares the received list of IdPs (if present) with the LEC IdP(s) retrieved from the selected cCard. If there is a non-empty intersection, then a cCard-specified IdP is contacted (this shall be the 'primary' IdP if possible); if not, then either the protocol terminates or the user could be asked to choose an IdP from amongst those in the SP list. The user could also be offered the choice to store the selected IdP (in the *Settings Store*) for future authentication attempts. If the SP does not specify a list of IdPs, then the cCard-associated IdP is contacted.

(f) IdP Auth → IdP: Authentication Request. The IdP Auth module issues an HTTP POST to submit a SOAP-based [27] Liberty authentication request message to the appropriate IdP. Note that this request must contain the authentication request as received from the SP.

11. Liberty IdP ⇌ User. If necessary, the IdP authenticates the user. Ideally this process would be mediated by the IDSpace system (using the *IdP Auth* module), in order to provide a user experience that is consistent across ID management systems. If successful, the IdP generates a SOAP-based, signed Liberty authentication response message and sends it to the IdP Auth module via an SSL/TLS channel.

12. Token Displayer ⇌ User. If the token is not end-to-end encrypted, the Token Displayer module displays the token and requests user consent to proceed. If consent is granted, the protocol continues; otherwise it terminates.

6.3 IDSpace and CardSpace

During or prior to use of IDSpace, the user must create a CardSpace-specific cCard (using the LIP) and/or import a CardSpace-managed InfoCard. The IDSpace generic protocol given in section 5.2, excluding step 12, could then be used to provide the functionality of CardSpace [10, 22].

7 Concluding Remarks

We have described an architecture for a client-based, platform-independent, protocol-agnostic ID management tool that operates in conjunction with a client web browser. A tool conforming to the architecture provides a user-intuitive means of managing digital identities and credentials for all user web activities.

7.1 Relationship to the Prior Art

7.1.1 CardSpace and Higgins

The Microsoft CardSpace system shares certain features in common with IDSpace. In particular, it too is client-based and operates in conjunction with a web browser. However, CardSpace requires the IdPs and SPs to implement a specific set of protocols for inter-communication (we refer to these as the 'CardSpace protocols', although many are based on WS-* standards). Although CardSpace supports a wide range of security token formats, these tokens must be sent using a very specific protocol suite.

This gives rise to a classic 'chicken and egg problem' — without an established identity infrastructure of IdPs, there is no (or little) incentive for SPs to make the changes necessary to support CardSpace. Similarly, without any customer SPs, there is no (or little) incentive to set up a CardSpace-specific IdP infrastructure.

By contrast, IDSpace gives the convenience and intuitive user experience of CardSpace, without requiring SPs and IdPs to change the way they work. That is, IDSpace enables convenient and more secure operation by end users, without any changes to the existing identity infrastructures or service providers. Moreover, once deployed, IDSpace will enable much simpler deployment of more sophisticated systems such as the CardSpace protocols (and the many other systems currently emerging).

The Higgins system (which originated with the goal of providing CardSpace-like functionality on non-Windows platforms) has somewhat similar objectives to IDSpace.

7.1.2 Other Schemes

In previous work [4, 5] we have described how to build browser extensions which enable CardSpace/Higgins selectors to support password management without requiring any changes to the SPs or to the identity selector. Operational, open-source prototypes[5] have also been described. These prototypes demonstrate the workability of certain aspects of the IDSpace system.

7.2 Novel Features

The main novel feature of IDSpace, as intimated above, is the proposal of an architecture for a client-based system which supports multiple ID management systems transparently to SPs and IdPs. That is, it combines the convenience and intuitiveness of the CardSpace user interface with support for multiple systems, without requiring any changes to existing SPs and IdPs. To our knowledge, the only previous work permitting client-based

[5] `http://iescripts.org/view-scripts-808p1.htm` and/or `http://sourceforge.net/projects/passcard/`

support for multiple ID management systems requires the SPs and IdPs to adopt new protocols.

The IDSpace architecture incorporates novel components, including the *Page Scanner*, *Activator*, *Identity System Selector* and *Token Displayer*, which are not found in the CardSpace or Higgins architectures. While much simpler versions of some of these novel components (notably the Page Scanner and Activator) have previously been described, [2, 4, 5, 6, 7], they have only been discussed in very specific contexts, and not in the general way in which they are used in IDSpace. Key elements of the architecture have been successfully prototyped.

7.3 Future Work

Our main initial goal is to complete an operational prototype of IDSpace, which we plan to make available for public scrutiny and testing. We intend that the initial version should support all the ID management systems discussed in this paper.

A variety of future directions for this research present themselves, a few of which we briefly mention.

- Apart from the ID management schemes mentioned previously, it would also be desirable if IDSpace could provide support for protocols providing a high degree of privacy protection for end users, notably U-Prove [11] and idemix [12]. This remains a topic of ongoing research.
- In previous work [3, 7], we have investigated using a client-based tool to support interoperation between different ID management systems, and a series of prototypes have been developed. It would be attractive (and straightforward) to build this functionality into an IDSpace implementation.
- Finally, in future work we intend to study variants of the architecture presented here to further enhance the security and privacy of user authorisation, whilst maintaining transparency to third parties.

References

[1] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, 2nd edition, 2002.

[2] H. S. Al-Sinani. Browser extension-based interoperation between OAuth and information card-based systems. Technical Report RHUL-MA-2011-15, Department of Mathematics, Royal Holloway, University of London, September 2011.

[3] H. S. Al-Sinani, W. A. Alrodhan, and C. J. Mitchell. CardSpace-Liberty integration for CardSpace users. In K. Klingenstein and C. M. Ellison, editors, *Proceedings of the 9th Symposium on Identity and Trust on the Internet, IDtrust 2010, Gaithersburg, Maryland, USA, April 13–15, 2010*, pages 12–25. ACM, New York, NY, 2010.

[4] H. S. Al-Sinani and C. J. Mitchell. Implementing PassCard — a CardSpace-based password manager. Technical Report RHUL-MA-2010-15, Department of Mathematics, Royal Holloway, University of London, December 2010.

[5] H. S. Al-Sinani and C. J. Mitchell. Using CardSpace as a password manager. In E. de Leeuw, S. Fischer-Huebner, and L. Fritsch, editors, *Policies and Research in Identity Management: Second IFIP WG 11.6 Working Conference, IDMAN 2010, Oslo, Norway, November 18–19, 2010, Proceedings*, volume 343 of *IFIP Advances in Information and Communication Technology*, pages 18–30. Springer, Boston, MA, 2010.

[6] H. S. Al-Sinani and C. J. Mitchell. Client-based CardSpace-OpenID interoperation. In E. Gelenbe, R. Lent, and G. Sakellari, editors, *Proceedings of ISCIS '11 — the 26th International Symposium on Computer and Information Sciences, 26–28 September 2011, London, UK*, Lecture Notes in Electrical Engineering (LNEE), pages 387–394. Springer, London, 2011. [Full version available at: `http://www.ma.rhul.ac.uk/techreports/2011/RHUL-MA-2011-12.pdf`].

[7] H. S. Al-Sinani and C. J. Mitchell. Client-based CardSpace-Shibboleth interoperation. Technical Report RHUL-MA-2011-13, Department of Mathematics, Royal Holloway, University of London, May 2011.

[8] W. Alrodhan. *Privacy and Practicality of Identity Management Systems: Academic Overview*. VDM Verlag Dr. Müller GmbH, Germany, 2011.

[9] E. Bertino and K. Takahashi. *Identity Management: Concepts, Technologies, and Systems*. Artech House Publishers, Norwood, MA, 2011.

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

[11] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, 2000.

[12] J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In V. Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications*

Security, CCS 2002, Washington, DC, USA, November 18-22, 2002, pages 21–30. ACM, New York, NY, 2002.

[13] M. Crowley. *Pro Internet Explorer 8 & 9 Development: Developing Powerful Applications for the Next Generation of IE*. Apress, New York, NY, 2010.

[14] E. Gallery. An overview of trusted computing technology. In C. J. Mitchell, editor, *Trusted Computing*, chapter 3, pages 29–114. IEE Press, London, 2005.

[15] C. Herley, P. C. van Oorschot, and A. S. Patrick. Passwords: If we're so smart, why are we still using them? In R. Dingledine and P. Golle, editors, *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23–26, 2009. Revised Selected Papers*, volume 5628 of *Lecture Notes in Computer Science*, pages 230–237. Springer-Verlag, Berlin, 2009.

[16] IETF. *Internet draft-ietf-oauth-v2-20: The OAuth 2.0 Authorization Protocol*, 2011.

[17] Internet2. *Shibboleth Architecture — Protocols and Profiles*, 2005.

[18] Internet2. *Shibboleth Architecture — Technical Overview*, 2005.

[19] J. Leach. Improving user security behaviour. *Computers & Security*, 22:685–692, 2003.

[20] Liberty Alliance Project. *Liberty ID-FF bindings and profiles specification*, 2004.

[21] Liberty Alliance Project. *Liberty ID-FF protocols and schema specification*, 2005.

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

[23] OASIS. *Identity Metasystem Interoperability Version 1.0 (IMI 1.0)*, 2009.

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

[25] L. M. Surhone, M. T. Timpledon, and S. F. Marsaken. *Security Assertion Markup Language: Security Domain, Single Sign-on, Identity Management, Access Control, OASIS, Liberty Alliance, SAML 1.1, SAML 2.0*. Betascript Publishing, 2010.

[26] L. M. Surhone, M. T. Timpledon, and S. F. Marseken, editors. *OpenID: Authentication, Login, Service, Digital Identity, Password, User, Software System, List of OpenID Providers, Yadis, Shared Secret*. Betascript Publishing, 2010.

[27] W3C. *W3C Recommendation: SOAP Version 1.2 Part 1: Messaging Framework*, 2007.

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

[29] P. J. Windley. *Digital Identity*. O'Reilly Media, Sebastopol, CA, 2005.