Using CardSpace as a Password-based Single Sign-on System

Haitham S. Al-Sinani and Chris J. Mitchell

Technical Report
RHUL–MA–2011–14
23 August 2011

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

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

Abstract

In this paper we propose a simple scheme that allows Card-Space to be used as a password-based single sign-on system, thereby both improving the usability and security of passwords as well as encouraging CardSpace adoption. We describe three related approaches to achieving password-based single sign-on using CardSpace. In each case users are able to store their credentials for a set of websites in a personal card, and use it to seamlessly single sign on to all these websites. The approaches do not require any changes to login servers or to the CardSpace identity selector and, in particular, they do not require websites to support CardSpace. We also describe three proof-of-concept prototypes and give usability, security and performance analyses.

Keywords: CardSpace, Single sign-on, Browser Extension

1 Introduction

Single sign-on (SSO) involves a user authenticating only once and thereby gaining access to multiple systems without the need to sign on separately at each of them. The notion of SSO is clearly attractive, not least from a user convenience perspective, particularly as the number of on-line services requiring authentication continues to grow. Furthermore, SSO helps to reduce the risk of exposure of passwords to malicious parties, including via key logging, shoulder surfing, etc. In this paper, we explore three different approaches to building a password-based SSO system on top of CardSpace.

CardSpace is a user-friendly tool supporting user authentication. To sign on to a website, a CardSpace user selects a virtual card, known as an information card (InfoCard), from an interface provided by the *CardSpace identity selector* (referred to below simply as the *selector*), instead of providing a username and password.

Despite the introduction of CardSpace (and other similar systems), the vast majority of websites still use username-password for authentication, and this is likely to continue for at least the next few years [9]. One major problem with CardSpace, and with other similar systems providing more secure means of user authentication, is that the transition from username-password is extremely difficult to achieve. Service providers will not wish to do the work necessary to support CardSpace if very few users employ it; equally, users are hardly likely to use CardSpace if it is only supported by a tiny minority of websites. The scheme proposed here is designed to help overcome this barrier to change by allowing an evolutionary deployment of CardSpace, initially as a password-based SSO system, and, subsequently,

once users are familiar with its interface, as a more sophisticated means of user authentication.

In this paper, we describe three simple approaches that all take advantage of the selector interface to offer SSO functionality. The goal is to develop a visual approach to SSO that is transparent to both the selector and relying parties (RPs). The techniques we discuss work with existing (unmodified) web servers, and, in particular, RPs are not required to support CardSpace. The scheme we describe here is related to a scheme called PassCard [3, 6] which allows CardSpace to be used as a password manager through the storage of user credentials in a CardSpace personal card. The main novel feature is the storage and subsequent use of multiple sets of credentials in a single InfoCard, allowing the provision of SSO functionality. An example use-case would involve a user storing the login credentials of their favourite (or most frequently-visited) websites, e.g. a university portal, G-mail/Hotmail, Facebook, YouTube and Twitter, in a single CardSpace personal card; selection of such a card will automatically sign on the user to all the relevant sites.

The remainder of the paper is organised as follows. Section 2 presents an overview of CardSpace, and section 3 presents the proposed scheme. In section 4 we describe three prototype implementations, and in section 5 we compare them. Section 6 underlines possible features and section 7 outlines potential issues. Section 8 highlights possible areas for related work, and, finally, section 9 concludes the paper.

2 CardSpace

2.1 Introduction

CardSpace is Microsoft's implementation of a digital identity metasystem, which 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 personal information from users, e.g. to support user authentication and authorisation.

The selector enables users to manage digital identities issued by a variety of identity providers (IdPs), and use them to access on-line services. Digital identities are visually represented to users as InfoCards, implemented as XML files that list the types of claim made by one party about itself or another party. Users can employ one (virtual) InfoCard to identify themselves to multiple websites. Alternatively, separate InfoCards can be used

in distinct situations. A website can specify which types of InfoCards are accepted and the types of claim which must be asserted.

There are two types of InfoCards: personal cards and managed cards. Personal cards are created by users themselves, and the claims listed in such an InfoCard are asserted by the self-issued identity provider (SIIP) that coexists with the selector on the user machine. In this paper we describe a way of using personal cards to enable CardSpace to function as a password-based SSO system. Managed cards, on the other hand, are obtained from remote IdPs.

InfoCards, personal or managed, do not contain any sensitive information; instead an InfoCard carries metadata that indicates the types of personal data associated with this identity, and from where assertions regarding this data can be obtained. The data referred to by a personal card is stored on the user machine, whereas the data referred to by a managed card is held by the IdP that issued it [1, 4, 7, 10, 12].

CardSpace is supported in Internet Explorer (IE) from version 7 onwards. Extensions to other browsers, e.g. Firefox[1] and Safari[2], also exist. An updated version, CardSpace 2.0 Beta 2, was released, although Microsoft announced in early 2011 that it will not ship; instead Microsoft has released a technology preview of U-Prove[3]. 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].

2.2 CardSpace Personal Cards

Since the proposed scheme builds on personal cards, we next outline their use. Prerequisites for use of a personal card include a CardSpace-enabled RP and a CardSpace-enabled user agent (UA), e.g. a web browser capable of invoking the selector. Personal cards can contain claims of the following 14 (editable) types: *First Name, Last Name, Email Address, Street, City, State, Postal Code, Country/Region, Home Phone, Other Phone, Mobile Phone, Date of Birth, Gender*, and *Web Page*.

When using personal cards, CardSpace adopts the following protocol. We describe the case where the RP does not employ a security token service (STS), a software component responsible for security policy and token

[1] `https://addons.mozilla.org/en-US/firefox/addon/cardspace-support-for-firefox/`

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

[3] `http://blogs.msdn.com/b/card/archive/2011/02/15/beyond-windows-cardspace.aspx`

management within an IdP and, optionally, within an RP.

1. UA $\rightarrow$ RP: HTTP/S Request (Get a login page).
2. RP $\rightarrow$ UA: HTTP/S Response. A login page is returned containing CardSpace-enabling tags in which the RP security policy is embedded.
3. User $\rightarrow$ UA: Selector Invocation. The UA offers the user the option to use CardSpace (e.g. via an RP-embedded logo on the login page), and selection of this option causes the agent to invoke the selector and pass it the RP policy. If this is the first time that this RP has been contacted, the selector will display the RP identity and give the user the option to either proceed or abort the protocol.
4. Selector $\rightarrow$ InfoCard: Displaying Cards. After evaluating the RP policy the selector highlights the 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.
5. User $\rightarrow$ Selector: Card Selection. The user chooses a personal card. (Alternatively, the user could create and choose a new personal card). At this point the user can check the requested claim types and decide whether or not to proceed. The selected InfoCard may contain several claims, but only the claims explicitly requested in the policy will be passed to the requesting RP.
6. Selector $\rightleftharpoons$ SIIP: Exchange of RST-RSTR. 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. Selector $\rightarrow$ UA $\rightarrow$ RP: RSTR. The RSTR is passed to the UA, which forwards it to the RP.
8. RP $\rightarrow$ User: Grant/Deny Access. The RP validates the received token, and, if satisfied, grants access.

The use of managed cards is covered in the relevant specifications [7, 10, 12, 11].

3 SingleSigner

We now describe the SSO scheme, which we call 'SingleSigner'. The idea behind SingleSigner is to store a set of user credentials in a special personal card, which, if selected, will transparently and automatically sign in

the user to a pre-defined set of websites. The parties involved are an RP, a CardSpace-enabled UA (e.g. a suitable web browser such as IE), and a browser extension installed on the user PC implementing the protocol described below.

Whenever a user visits a website requiring username-password authentication, the SingleSigner functionality can be invoked by clicking on a special icon added to the website by SingleSigner. This causes the selector to run, at which point the user must select a special personal card containing the credentials for the visited site (encoded in a SingleSigner-specific format). The user will be automatically logged in to the visited site and also to all the other sites whose credentials are stored in the selected card.

The version of the system described in section 3.2 and the prototypes described in section 4 only work if the visited site does not use HTTPS[4]. This limitation is shared by the original version of PassCard [3]. However, as discussed in section 7, this limitation can be removed in exactly the same way as it is in the current version of PassCard [2, 6].

3.1 Prerequisites

The scheme has the following operational requirements.

- Either prior to or during use of the scheme, the user must create a special personal card, referred to as an SSOcard, containing the (URL, username, password) triples for the websites supported by this card. These triples must be stored using a specific encoding in pre-defined card fields[5]. For ease of identification, the user can give the SSOcard a meaningful name, e.g. of the corresponding websites. The user can also upload an image for the SSOcard, e.g. containing the logos of the sites whose credentials it contains.

- A special browser extension must be installed on the user PC. This must be able to implement the protocol described in section 3.2, including reading and modifying browser-rendered web pages, reading CardSpace-issued RSTR tokens, and adding a special icon to RP web pages[6] to enable the user to invoke the scheme. To maximise acceptability, a user should also be able to enable or disable it at will.

[4]Note that only the visited site must not use HTTPS; other sites included on the same SSOcard can use either HTTPS or HTTP (see section 7).

[5]The credential sets could alternatively be stored in a single card field, separated using a special character. However, this might make using the scheme more difficult for the end user.

[6]Regardless of whether or not an RP already supports CardSpace, the browser exten-

3.2 Protocol Flow

The protocol steps are as follows; a summary of the protocol is shown in Fig. 1.

1. UA → RP: HTTP GET Request (a login page is requested).
2. RP → UA: HTTP Response (the login page is returned).
3. Browser extension→ UA: Preprocessing. The browser extension performs the following processes using the login page provided by the RP.
   (a) It scans the page for a login form containing username and password fields and a submit button. If all are found, it continues; if not it aborts.
   (b) It adds CardSpace-enabling tags to the login page, including embedding a security policy. The embedded policy must request all the card fields used by the implementation of SingleSigner, where the (editable) fields must be marked as optional. If all fields were marked as mandatory then only those cards containing data in every SSOcard field would be highlighted by the selector.
   (c) It adds a function to the login page to intercept the RSTR token that will later be returned by the selector.
   (d) It causes a special icon to appear above the submit button, in such a way that clicking it invokes the selector.
4. User → UA: Icon Clicking. The user clicks on the added icon and the selector lights up.
5. User → Selector: Card Selection. The user selects and submits an SSOcard. Alternatively, the user could create and choose a new SSOcard. The selector creates and sends an RST to the SIIP, which responds with an RSTR.
6. Selector → UA: RSTR. The selector passes the RSTR to the UA.
7. Browser Extension [Intercepts] RSTR. The browser extension performs the following tasks.

sion will always add the special icon to the RP page as long as it detects username-password prompts on the page. Informal tests on the prototype implementations suggest that this will not disrupt the normal operation of CardSpace.

(a) It intercepts and parses the RSTR.

(b) It extracts the URL for the visited site together with the username and password associated with this URL from one of the pre-specified fields.

(c) It auto-populates and auto-submits the login form using the extracted username and password.

(d) The website server verifies the credentials it receives, and, if satisfied, grants access.

8. Browser Extension [Performs] SSO. The browser extension repeats steps 7b–7d for every other website included in the user-selected SSOcard, invoking a new browser window for each site[7]. Note that the detailed operation of this step will vary depending on the method being used (see below).

There are a variety of ways in which the user credentials could be sent to a website in step 8. We next discuss three possible approaches to achieving this, namely: URL query parameters, client-side cookies, and hidden html form variables. Note that the choice of approach only affects step 8. As discussed in sections 4.4.3 and 5, the URL query parameters approach is more usable than the other two approaches, and is hence described first as the primary approach.

3.2.1 Primary Approach

For each site listed in the SSOcard, the browser extension creates a URL containing the site's address and the user credentials for this site (as taken from the SSOcard). The browser extension then creates a browser window for each site. Finally, the browser extension reads the credentials from the URL, and auto-populates and submits the login form.

The fact that SingleSigner automatically creates a browser window for each site included in an SSOcard could be somewhat intrusive for the user[8]. We therefore also propose a slightly different method of operation. This

[7] A new browser window is invoked in order to maintain the established authenticated session with each of the websites. Following a successful login process, most websites typically create a short-lived cookie (a session cookie — see `http://en.wikipedia.org/wiki/HTTP_cookie`) which will be deleted if the browser window is closed or if a certain period of inactivity elapses.

[8] Whether or not this is a problem in practice depends partly on the number of sets of credentials included in a single SSOcard.

alternative method operates as in step 8, except that, after invoking a new browser window for each site in the SSOcard, the browser extension stores (e.g. in a cookie) the URL of the page to which the user is granted access following a successful authentication. The browser extension then attempts to close (but not sign out) each page it has invoked; the user-visited page will, of course, remain open since it was not invoked by the extension. When a user later visits a website included in the submitted SSOcard, the browser extension will auto-redirect the user to the logged-in page that the extension stored earlier[9] and then terminate. As long as the main browser session is still live, the user logged-in session at each site included in the SSOcard should still be valid; however, the session may be invalid if the main browser session is closed or if a certain period of inactivity (as determined by the site server) has elapsed. This method has been successfully tested with the URL query parameters approach.

3.2.2 Other Approaches

We next describe two other approaches to implementing step 8 of the protocol, namely the use of either cookies or hidden form variables.

Cookies. From a user perspective this approach is similar to the URL query parameters approach, except that the user must append a 'flag' word to each credential triple when the SSOcard is created; this word must be manually removed once the SSOcard has been used.

The browser extension first examines the RSTR message to detect if the flag word is present at the end of each set of credentials; if so, it runs in the exact same way as the URL query parameters protocol, except that before the browser extension auto-populates and submits the login form, it sets a persistent cookie[10] in order to store the username and password values for future logins. A cookie is thus created for each website whose credentials are stored in the SSOcard. If the flag word is not present, then the extension invokes a browser window for each site whose credentials are included in the SSOcard. It then recovers

[9] If such a page was not stored, then some website servers would prompt the user to re-authenticate.

[10] Persistent cookies can survive across a number of sessions, including after exiting the browser and/or after a machine reboot. Such cookies have an expiry date; if a cookie expires it is deleted.

the user credentials from the appropriate cookie[11], and uses them to auto-populate the site login form, which it auto-submits.

Note that, unlike the other approaches, here the user credentials are not retrieved from a hidden HTML form or from a URL, thereby reducing the exposure of username-password values (since the values will not be shown in the browser URL address). This provides protection against shoulder-surfing attacks.

Hidden Form Variables. In this approach the browser extension creates a separate invisible HTML form (containing hidden variables) for each site listed in the SSOcard. Each form is auto-filled using the user credentials and then auto-submitted. The extension opens a new browser window for each site contained in the SSOcard.

To make this approach work, certain RP-specific information (notably the URL of the login server and the names given to the username and password fields) must be available to the browser extension independently of the SSOcard. This means that every time a new SSOcard is created (or an existing SSOcard is modified to include a new credential set) the browser extension must be modified to incorporate this information.

4 Implementation

We now describe three proof-of-concept prototypes implementing the SingleSigner scheme presented in section 3.2, one for each of the three described approaches to realising step 8. The prototypes operate with both the CardSpace and the Higgins[12] identity selectors without any modification.

4.1 Shared Properties

The prototypes are all coded as JavaScript [15] plug-ins, chosen because its wide adoption should simplify the porting task for other browsers. They use the Document Object Model (DOM) to inspect and manipulate HTML

[11] If the cookie expires or is removed, the browser extension will fail to find an appropriate cookie and will then prompt the user to add the flag word to the end of each set of credentials in the relevant SSOcard.

[12] `http://wiki.eclipse.org/GTK_Selector_1.1-Win`

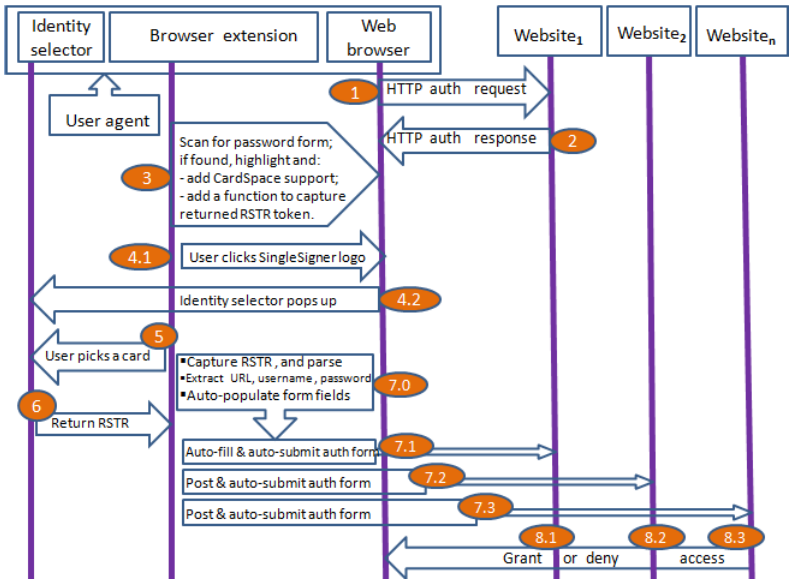

Figure 1: Protocol Flow

pages and XML documents. The JavaScript code is executed using a C#-driven browser helper object (BHO), a DLL (Dynamic-Link Library) 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. In each case, SingleSigner can readily be enabled or disabled using the add-on manager in the IE 'Tools' menu.

4.1.1 SSOcard Format.

The prototype permits credential sets to be stored in any of the 14 personal card fields with the exception of the *Birthday* and *Gender* fields (which cannot contain arbitrary strings). Credential sets must be stored in the format:

<URL> <username> <password>

where the three fields are separated by a single space character.

4.1.2 Operation.

In step 3 of the protocol (see section 3.2) the plug-in processes the RP web page in the following way.

3.1 It scans the web page for a login form containing a pair of username and password fields and a submit button. More specifically, the following procedure is used.

(a) It scans the web page for a form tag.

(b) If a form tag is found, it searches the form for three input tags referring to username, password, and submit, as follows:

(i) it searches for an input tag of type 'text';

(ii) if found, it searches for another input tag of type 'password'; and

(iii) if found, it searches for another input tag of type 'submit'; if not found, it searches for an input tag of type 'image' and, if unsuccessful, searches for an event-based input tag of type 'button'.

(c) If all three fields are detected, then it highlights the username and password fields in green for ease of identification[13].

The above process involves the following detailed processing.

- Scanning does not terminate successfully unless both the username and password fields and the submit button have been detected in a single form, as a web page could potentially contain more than one input tag of type 'text', such as those used for searching.
- To differentiate between registration and login web pages, the plug-in terminates if it detects more than one password field between the username and submit fields. Whereas it appears common for a login page to only have a single password field before the submit button, registration pages typically have two password fields (before the submit button): the first for the user to enter their password, and the second to confirm their password. Examples include the registration and login pages hosted by major websites such as Google, YouTube, Yahoo, Microsoft Research, SpringerLink[14], etc.
- When searching for the form submission button, if no submitting input tag is found then the plug-in searches for an 'image' tag.

[13] A potential advantage of this step is that if the wrong fields are highlighted, then the user will know that the scheme should not be used.

[14] Websites most recently checked on 24/11/2010.

This is because, instead of a submit button, some websites display a clickable image[15] with similar functionality.

- Whereas it appears common for a username field to be immediately followed by a password field, a submit button may not always immediately follow a password field. For example, some major websites (such as Google, YouTube, Yahoo, Gmail, SpringerLink) add a 'Stay signed in' or 'Remember me' check box between the password field and the submit button[16]. The plug-in addresses this issue by skipping all tags between the password field and the submit button, including those of type 'checkbox'.

3.2 It adds an HTML object tag that allows the user to invoke the selector. Within the object tag, it sets the param tags to indicate that the RP security policy requires SSOcards to contain at least one (compulsory) field[17], namely the *First Name* field, and to also include 11 (optional) fields, namely *Last Name, Email Address, Street, City, State, Postal Code, Country/Region, Home Phone, Other Phone, Mobile Phone*, and *Web Page*.

3.3 It adds a function to the head section of the RP login page to intercept the XML-based RSTR message returned by the selector.

3.4 It inserts the SingleSigner logo (see Fig. 2) in the login page, in such a way that it appears just before the submit button. The logo is associated with an 'on-click' event, so that, if clicked, the selector is invoked (after calling the added function). To cater for users with sight difficulties or web browsers configured not to display images, a text field can replace the logo. This text is also displayed when the mouse is held over the SingleSigner logo, indicating that SingleSigner can be used to sign on.

In step 7, the plug-in performs the following steps.

[15] This includes an image tag embedded in a hyperlink (anchor) tag, an image tag on its own, an image tag embedded inside a button tag, an event-based button tag, etc.

[16] Websites most recently checked on 24/11/2010.

[17] From a user perspective, marking at least one field as mandatory means less computation (as explained below) and, ultimately, a faster authentication process, hence helping user acceptability. In the SSOcard selection step, if the SSOcard only contains one credential set in a mandatory field then the user only needs to choose an SSOcard and click the 'Send' button. If the field was not mandatory but instead optional, then the user would first need to tick the optional field before clicking the 'Send' button. From an operational perspective, an RP security policy must contain at least one required/mandatory claim; of course this claim could be the *Personal Private Identifier* (PPID) claim [7].

Figure 2: SingleSigner Logo

7.1 It intercepts the XML-based RSTR message using the added function.

7.2 It parses the intercepted token and extracts the value of the *First Name* field as well as the values of any other optional fields present, thus learning the set of websites supported by the SSOcard.

The plug-in uses an XML parser built into the browser to read and manipulate the intercepted XML token. The plug-in passes the token to the parser, which reads it and converts it into an XML DOM object that can be accessed and manipulated by JavaScript.

7.3 It auto-fills the username and password fields and auto-submits the login form of the currently visited website[18] using the JavaScript 'click()' method.

4.2 The URL Query Parameters Prototype

4.2.1 Operation.

Steps 3.1–7.3 are precisely the same as those described in section 4.1.2.

7.4 For each other site included in the SSOcard:

(a) the plug-in invokes a new browser window (using the JavaScript built-in method 'open.window()'), that retrieves the site's login-page (using the URL provided in the SSOcard). Note that it sends the username-password values embedded in the URL (i.e. URL query parameters) as part of this HTTP request; and

(b) using the login-page returned in the previous step, the plug-in:

[18]It detects the correct username-password values for the visited site by comparing its domain name with the URLs contained in the 'credential' fields of the RSTR message.

(i) parses the URL query parameters to obtain the values of the username and password;

(ii) locates the username, password, and submit fields;

(iii) auto-populates the username and password fields with the username and password values; and

(iv) auto-submits the login form using the JavaScript 'click' method.

4.2.2 Protecting Credentials.

This approach involves embedding the username and password in the URL. Thus, if they are embedded in clear text, they will be vulnerable to shoulder-surfing attacks. To address this potential problem, in step 7.4a the prototype encrypts these values using AES in CBC mode [13], and decrypts them in step 7.4b.

The AES key used for username/password encryption is stored in the plug-in. This would be a security issue if the same key was used by every copy of the plug-in, but a unique random key could be generated at the time the plug-in is installed. The presence of the key on the PC does not significantly increase the risks to credential secrecy, since the credentials must in any event be stored on the PC.

Embedding the credential values in a URL could also cause problems because of the URL size limitations; however, such drawbacks are not likely to be a major problem here since the amount of data involved is relatively small (see [2]). The use of cryptography also results in a slight performance delay.

4.3 The Cookies Prototype

4.3.1 SSOcard Format.

The format of the SSOcard is identical to that described in section 4.1.1, except that, at the time of card creation, a flag word must be appended to each credential triple (see also section 3.2.2). The prototype expects to find a string of the form 'cookie9', where '9' indicates the lifetime (in days) of the persistent cookie created by the plug-in. After first use, this flag word must be removed by the user (and added back if the credential cookie expires).

4.3.2 Operation.

Steps 3.1–7.3 are precisely the same as those described in section 4.1.2. Step 7.4 is precisely as in section 4.2.1, except that the following step is added

between steps 7.4.b.iii and 7.4.b.iv:

- the plug-in (as described in section 3.2.2) first examines the RSTR for the flag word; if present it creates a persistent cookie containing the username and password. If it is not present then it recovers the username and password from the cookie.

4.3.3 Protecting Credentials.

This approach involves embedding the username and password in a cookie. Thus, if they are stored in clear text, they will be readable by anyone with temporary access to the machine. As in section 4.2.2, this threat can be reduced by encrypting the data in the cookie, e.g. using a key only known to the browser plug-in.

4.4 The Hidden Form Fields Prototype

4.4.1 Initialisation.

In this approach, the user must make certain modifications to the plug-in source code. The user must first obtain the URL of the login server for each website included in an SSOcard; this can be found by viewing the login-page HTML source and retrieving the 'action' URL of the login form. The user must also obtain the 'names' given to the username and password input fields[19], which could also be found from the page HTML source. The user must then insert the URL and the names of the username and password fields into the plug-in source code, following the plug-in instructions.

4.4.2 Operation.

Steps 3.1–7.3 are precisely the same as those described in section 4.1.2.

7.4 For each other site included in the SSOcard, it:

(a) creates an invisible HTML form containing at least two hidden 'input' variables, and then auto-fills each variable with the corresponding username or password;

(b) creates a new browser window using the JavaScript built-in method 'open.window()'; and

(c) auto-submits the invisible HTML form.

[19] This is important since the site's login server will use these names to retrieve username-password values from the HTTP POST array.

4.4.3 Operational Issues.

Protoype testing reveals that some website login servers impose restrictions on externally posted/submitted forms for security reasons. That is, if a user is currently visiting site `a.com` and the browser plug-in submits/posts a login form to site `b.com`, then access to a protected resource in domain `b.com` will not be granted even if the user credentials are correct.

5 Comparison

We next compare the three approaches in terms of usability, security, and performance.

- **Usability.** The primary approach only requires entry of (username, password, URL) triples into SSOcards, and hence is clearly the most usable. The other two methods either require manual modifications to the plug-in source code (likely to be completely beyond most users) or the additional overhead of adding/removing flag words to/from SSO-card entries.

- **Security.** The primary approach involves adding encrypted credentials to the URL; recovering the encryption key from the browser plug-in may be slightly simpler than recovering the credentials from the CardSpace store. The cookies approach reduces credential exposure to first card use, and the hidden form variables approach avoids this risk.

- **Performance.** The hidden forms approach has the advantage that, once the browser windows are opened, no further processing is required. Less processing is required for the primary approach than the cookies approach.

Table 1 below summarises the comparison.

Table 1: Comparison

	URL Params	Cookies	Hidden Forms
Usability	++	+	-
Security	+	++	+++
Performance	++	+	+++

6 Features of SingleSigner

6.1 Security.

SingleSigner uses the functionality of the CardSpace identity selector, and is supported by its built-in security features. For example, the selector runs in a separate private desktop session, mitigating the risk of other applications, e.g. malware, from interacting or interfering with it. In addition, all values inserted in the fields of an SSOcard are stored in encrypted form on the user machine.

Furthermore, the selector identifies the visited 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[20]. This helps to support mutual authentication, since the user and the RP are both identified to each other.

Unlike many other SSO systems, SingleSigner avoids the need for trusted third parties. In addition, the automatic form-filling feature reduces exposure to shoulder-surfing attacks and also helps to thwart key loggers.

SingleSigner helps to mitigate the threat of phishing attacks by only passing a user password to the corresponding URL in the SSOcard. SingleSigner also helps to support the use of strong site-unique passwords, since users do not need to memorise them. In addition, SingleSigner makes the periodic update/change of passwords easier and more often.

Finally, compromise of any one password does not threaten the confidentiality of other passwords, or compromise user authentication at other sites (as would be the case, for example, if a password for an OpenID/Liberty identity provider was compromised).

6.2 Usability.

We now consider the usability features of SingleSigner.

- **User Experience.** SingleSigner provides a simple, intuitive and consistent user experience through its use of the CardSpace identity selector interface. At the same time, it familiarises users with CardSpace, thereby potentially facilitating future adoption of more secure means of authentication. SingleSigner credentials are stored in SSOcards which can be equipped with a readily recognisable image, and users can access their favourite set of password-protected websites using a single

[20]This enhances security by comparison with 'conventional' password-based authentication, where the RP is not identified to the user.

card. In addition, the automatic form-filling feature reduces the time and effort required for username-password authentication.

- **Transparency.** SingleSigner operates transparently to external parties, and hence does not require any changes to RPs or to identity selectors. In particular, it does not require websites to support CardSpace.

- **Flexibility.** SingleSigner is flexible, since users can choose whether or not to use it simply by electing to click the SingleSigner logo (or not). In addition, each SSOcard can be used with any site included in the SSOcard. SingleSigner can also be used with multiple identity selectors, including those provided by Microsoft CardSpace and Eclipse Higgins.

- **Availability.** SingleSigner does not possess any single points of failure; failing to access a site whose credentials are included in an SSOcard will not impede access to other sites covered by the same card.

- **Costs.** The SSO functionality is likely to reduce organisational IT costs as a result of fewer password-related help desk calls.

- **Roaming.** By making use of CardSpace features, SingleSigner supports a degree of roaming. A user can transfer SSOcards from one PC to another using the CardSpace backup facilities. Indeed, if the CardSpace backup file (which holds data in encrypted form) is stored on a portable storage medium (e.g. a USB drive) then full mobility is provided, as well as robustness in the form of protection against loss of credential data.

7 Potential Issues

Perhaps the most obvious limitation of the SingleSigner system is that anyone with access to a Windows user account can access the SSOcards and use the stored credentials. This is a fundamental limitation of CardSpace itself which, by default, does not impose any additional password protection on the use of the selector. To address this issue, we observe that CardSpace allows individual InfoCards to be PIN-protected, which should be seriously considered for SSOcards stored on machines accessible to other users. In addition, it may be possible to cause CardSpace to run under UAC (User Account Control), so that running CardSpace causes Windows to prompt

the user for an administrator password. Moreover, CardSpace-based authentication could be enhanced by using a mobile device. During the process of user authentication on a PC using CardSpace, a random and short-lived one-time password is sent to the user's mobile device; this must then be entered into the PC by the user when prompted [5].

The version of the SingleSigner prototype described in this paper does not work as intended, if the website at which the initial authentication takes place uses HTTPS. This is because, if such a website has a certificate, then the selector will, by default, encrypt the SAML-based RSTR message using the public key of the requesting site. The plug-in does not have access to the site's private key, and hence will be unable to decrypt the token. As a result, SingleSigner will not be able to perform automatic form-filling as it cannot obtain the username/password values. However, if the site at which the initial authentication takes place uses HTTP, then SingleSigner will work as intended even if all other sites included on the same SSOcard use HTTPS.

A more satisfactory solution to this issue would be to configure SingleSigner to redirect the user to an (arbitrary) HTTP-based website if the first site uses HTTPS (as discussed in [2]). In this case, SingleSigner could read the SAML token returned from the selector as it would not be encrypted. SingleSigner could then submit the credentials automatically to the target HTTPS-based site using any of the three discussed methods.

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

Use of SSO systems in general, including SingleSigner, may lead to privacy violations. User interactions on the web could be linked to build a unique user profile. For example, the HTTP referrer field and cookies might help to build such a profile. Whilst it is common for the profile builders to insist that the profiles are only used for advertising purposes, such actions, particularly when conducted in the absence of informed user consent, remains a likely threat. However, modern browsers help to minimise such a threat.

Finally, older browsers (or browsers with scripting disabled) may not be able to run SingleSigner, as it was built using JavaScript. However, most modern browsers support JavaScript (or ECMAscript), and so this seems unlikely to be a major usability obstacle.

8 Related Work

A very wide range of Internet SSO schemes[21] have been proposed [8, 14]. We observe that, using the taxonomy of [14], SingleSigner is a local pseudo-SSO scheme in that the credentials are stored locally and the RPs are not aware of the operation of the scheme. Other examples of such schemes include Novell's SecureLogin[22], Passlogix V-GO[23] and Protocom's SecureLogin[24]. Automatic form-fillers, e.g. the automatic form completion functions of popular web browsers such as IE and Firefox, can also be regarded as local pseudo-SSO schemes [14].

Perhaps the most distinctive feature of SingleSigner is its dependence on CardSpace, whereas other SSO systems are independent applications. SingleSigner can thus benefit from the CardSpace security features, which may give users greater confidence in its use.

9 Conclusions and Future Work

In this paper we have proposed SingleSigner, a simple scheme that allows CardSpace to be used as a password-based single sign-on system. Three related approaches to achieving password-based single sign-on using CardSpace were discussed. In each case users are able to store their credentials for a set of websites in a personal card, and use it to seamlessly single sign on to them all. The approaches do not require any changes to login servers or to the CardSpace identity selector and, in particular, they do not require websites to support CardSpace.

The schemes use the CardSpace identity selector to seamlessly single sign on users to password-protected websites. It extends the use of personal cards to allow for transparent password-based single sign-on, thereby both improving the usability and security of passwords as well as encouraging CardSpace adoption.

Planned future work includes building a portable version of the CardSpace password-based single sign-on system to help roaming users who might not have installation privileges or might not be able to use their personal machines, e.g. in Internet cafes, airport lounges, etc. In addition, we plan to investigate the possibility of extending the proposed scheme to support single sign-off.

[21] `http://en.wikipedia.org/wiki/List_of_single_sign-on_implementations`
[22] `http://www.novell.com/products/securelogin/`
[23] `http://www.passlogix.com/sso`
[24] `http://www.protocom.cc`

Acknowledgements

The first author is sponsored by the Diwan of Royal Court, Sultanate of Oman.

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 the 9th Symposium on Identity and Trust on the Internet, (IDtrust'10), Gaithersburg, Maryland, USA, April 13–15, 2010*, pages 12–25. ACM, New York, 2010.

[2] Haitham S. Al-Sinani and Chris J. Mitchell. *Implementing PassCard — a CardSpace-based Password Manager*. Technical Report: RHUL–MA–2010–15 (Department of Mathematics, Royal Holloway, University of London), 2010. `http://www.ma.rhul.ac.uk/static/techrep/2010/RHUL-MA-2010-15.pdf`.

[3] Haitham S. Al-Sinani and Chris J. Mitchell. Using CardSpace as a password manager. In Elisabeth de Leeuw, Simone Fischer-Hübner, and Lothar Fritsch, editors, *Proceedings of IFIP IDMAN 2010 — 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, November 18–19 2010, Oslo, Norway*, volume 343 of *IFIP Advances in Information and Communication Technology*, pages 18–30. Springer, Boston, 2010.

[4] Haitham S. Al-Sinani and Chris J. Mitchell. Client-based CardSpace-OpenID interoperation. In *Proceedings of ISCIS 2011 — 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. [See also the full version at: Royal Holloway, University of London, Mathematics Department Technical Report RHUL-MA-2011–12, May 2011, 23 pages, `http://www.ma.rhul.ac.uk/techreports/2011/RHUL-MA-2011-12.pdf`].

[5] Haitham S. Al-Sinani and Chris J. Mitchell. Enhancing CardSpace authentication using a mobile device. In Yingjiu Li, editor, *Proceedings of the 25th IFIP WG 11.3 Conference on Data and Applications Se-*

curity and Privacy (DBSEC'11), Richmond, Virginia, USA, 11-13 of July 2011, volume 6818, pages 201–216. Springer (LNCS), 2011.

[6] Haitham S. Al-Sinani and Chris J. Mitchell. Extending the scope of cardspace. In *Proceedings of the 4th International Conference on Security of Information and Networks (SIN'11), 14–19 of November 2011, Sydney, Australia (to appear)*. To be published in ACM, 2011.

[7] 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.

[8] Jan De Clercq. Single sign-on architectures. In George I. Davida, Yair Frankel, and Owen Rees, editors, *Proceedings of Infrastructure Security, International Conference, InfraSec'02, Bristol, UK, October 1–3, 2002*, volume 2437 of *Lecture Notes in Computer Science*, pages 40–58. Springer-Verlag, Berlin, Heidelberg, 2002.

[9] Cormac Herley, Paul C. van Oorschot, and Andrew S. Patrick. Passwords: If we're so smart, why are we still using them? In Roger Dingledine and Philippe 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*. Springer-Verlag, Berlin, Heidelberg, 230–237, 2009.

[10] Michael B. Jones. *A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers*. Microsoft Corporation, 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] Marc Mercuri. *Beginning Information Cards and CardSpace: From Novice to Professional*. Apress, New York, 2007.

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

[14] Andreas Pashalidis and Chris J. Mitchell. A taxonomy of single sign-on systems. In Rei Safavi-Naini and Jennifer Seberry, editors, *ACISP'03:*

Proceedings of the 8th Australasian conference on Information security and privacy, Wollongong, Australia, July 9–11 2003, volume 2727 of *Lecture Notes in Computer Science*, pages 249–264. Springer-Verlag, Berlin, Heidelberg, 2003.

[15] Thomas A. Powell and Fritz Schneider. *Javascript: The Complete Reference*. McGraw-Hill Osborne Media, Berkeley, CA, 2nd edition, 2004.