Implementing PassCard — a CardSpace-based Password Manager

Haitham S. Al-Sinani and Chris J. Mitchell

Technical Report
RHUL–MA–2010–15
20 December 2010

Royal Holloway
University of London

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

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

Abstract

The recently-proposed PassCard scheme enables CardSpace to be used as a password manager, thereby both improving the usability and security of passwords as well as encouraging CardSpace adoption. However, this scheme does not work with websites using HTTPS, seriously limiting its practicality. In this paper we extend PassCard to support sites using both HTTP and HTTPS. Usernames and passwords are stored in CardSpace personal cards, and these cards can be used to sign on transparently to corresponding websites. PassCard does not require any changes to login servers, default browser security settings or to the CardSpace identity selector and, in particular, it does not require websites to support CardSpace. We describe how this new version of PassCard operates, and present a proof-of-concept prototype. Security, usability and operational analyses are also provided.

Keywords: CardSpace, PassCard, Password Manager, Browser Extension

1 Introduction

The most common means of user authentication remains the use of passwords, despite their well-known shortcomings. Moreover, as the number of on-line services requiring passwords continues to grow, users increasingly re-use passwords, write them down in insecure ways, and/or employ passwords which can be readily guessed. The result is an ever-increasing risk of exposure of passwords to malicious parties. Passwords could also be stolen [3, 7] through key logging, phishing, sniffing, shoulder surfing, etc.

A solution that enables the use of site-unique strong passwords whilst maintaining user security and privacy is thus needed [4]. Password managers of various types have been proposed to meet this need. A password manager stores usernames and passwords and makes them available when required. Users are not required to remember any passwords apart from a single master password which can be used to lock/un-lock the password manager. Password managers can be particularly helpful when a relatively large number of passwords are required to access multiple on-line services. Password managers can be seen as potential alternatives to single sign-on systems such as Windows Live ID, formally known as Passport[1], Liberty Alliance Project[2] [15], and OpenID[3].

[1] http://passport.net/

[2] http://www.projectliberty.org/

[3] http://openid.net/

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 (CIdS), 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 [7]. 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. PassCard is designed to help overcome this barrier to change by allowing an evolutionary deployment of CardSpace, initially as a password manager and subsequently, once users are familiar with its interface, as a more sophisticated means of user authentication.

In this paper we propose a new version of PassCard [1] that uses the CIdS as a password manager. The goal is to develop a simple and intuitive approach to password management, transparent to both the CIdS and relying parties (RPs). Unlike the previous version of PassCard [1], which only supports sites using HTTP, the new version supports both HTTP and HTTPS-enabled sites.

The remainder of the paper is organised as follows. Section 2 presents an overview of CardSpace, and, in section 3, we describe the proposed scheme. We describe a prototype implementation in section 4, and, in section 5, we outline a number of PassCard features and limitations. Section 6 reviews related work, and, finally, section 7 concludes the paper.

2 CardSpace

2.1 Introduction

Microsoft CardSpace is an identity management system that 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 CIdS 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, which are 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. Websites can request different types of cards and/or different types of claims.

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 (SIP) that coexists with the CIdS on the user machine. In this paper we take advantage of such personal cards to enable CardSpace to function as a password manager. Managed cards, on the other hand, are obtained from remote IdPs.

The InfoCards themselves 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 [2, 8, 11].

By default, CardSpace is supported in Internet Explorer (IE) from version 7 onwards. Extensions to other browsers, such as Firefox[4] and Safari[5], also exist. Microsoft has recently released an updated version of CardSpace, known as Windows CardSpace 2.0 Beta 2[6]. However, in this paper we refer throughout to the CardSpace version that is shipped by default as part of Windows Vista and Windows 7, which has also been approved as an OASIS standard [9].

2.2 CardSpace Personal Cards

Since PassCard 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, e.g. a web browser capable of invoking the CIdS. 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 assume that the RP does not employ a security token service (STS)[7].

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

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

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

[7] An STS is a software component responsible for security policy and token management

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

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

3. The user agent offers the user the option to use CardSpace (e.g. via a button on the RP page), and selection of this option causes the agent to invoke the CIdS and pass it the RP policy. If this is the first time that this RP has been contacted, the CIdS will display the RP identity and give the user the option to either proceed or abort the protocol.

4. After evaluating the RP policy the CIdS 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. 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. The CIdS creates and sends a SAML-based request security token (RST) to the SIP, which responds with a SAML-based request security token response (RSTR).

7. The RSTR is passed to the user agent, which forwards it to the RP.

8. The RP validates the received token, and, if satisfied, grants access.

Note that if the visited site uses HTTPS, then the RSTR message in step 7 (above) will be encrypted using the public key of the requesting RP. This explains why the previous version of PassCard, which relies on a browser extension that parses the RSTR, only works with websites using HTTP. The extension does not have access to the key necessary to decrypt the token, and hence cannot operate. This issue is addressed in the revised PassCard scheme described in section 3 below.

Details of how CardSpace managed cards are used are given in the relevant specifications [2, 8, 9, 11].

within an IdP and, optionally, within an RP.

3 PassCard

We now describe the operation of the new version of PassCard. The parties involved are a CardSpace-enabled RP, a CardSpace-enabled user agent (e.g. a suitable web browser), an HTTP Server (HS), and a browser extension implementing the protocol described in 3.2. The introduction of the HS (see also section 4.3.2 below) addresses the main limitation with the previous version of PassCard (described at the end of section 2.2), thus enabling PassCard to also operate with HTTPS-based websites.

3.1 Prerequisites

The scheme has the following operational requirements.

- Either prior to, or during, use of the scheme, the user must first create a special personal card, referred to here as a PassCard, containing a username and password in certain card fields, the choice of which is implementation-specific. Basic protection against phishing can be provided if the URL of the target website is included in the PassCard. However, this is optional, as users may wish to use a single PassCard with multiple websites sharing the same user credentials.
- As in the previous version [1], PassCard is based on a browser extension which must be able to read, inspect and modify browser-rendered web pages, and must also be able to read CardSpace-issued RSTR tokens. In addition, it must be able to automatically invoke the CIdS, fill in and submit login forms, and forward username-password values (via HTTP redirects) to RPs. Furthermore, it must be able to start automatically and be enabled or disabled by the user. Finally, the browser extension must be able to add a clickable PassCard logo (see Fig. 3), a modified version of the CardSpace logo, to the RP web page. This enables the user to invoke the CIdS and to subsequently select (or create) a PassCard.

3.2 PassCard Operation

We now describe the protocol steps for PassCard. Its operation differs depending on whether the RP uses HTTP or HTTPS. We therefore divide the protocol description into two cases. Protocol steps 1–3c are the same for both cases, and hence we describe these steps first. Note that the HTTP case is precisely the same as the previous version of PassCard [1]; we include the description here for completeness.

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

2. RP → user agent. HTTP response: (the login page is returned).

3. 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 a username and password field and a submit button.

   (b) If all are found, it highlights the username and password fields.

   (c) It determines the communication protocol (i.e. HTTP or HTTPS) in use.

Execution continues as described in sections 3.2.1 and 3.2.2, depending on whether the RP is using HTTP or HTTPS, respectively.

3.2.1 HTTP-based PassCard

Steps 3d–8 below apply in the case where the RP uses HTTP.

3. Following step 3c in section 3.2, the extension continues as follows.

   (d) It adds CardSpace-enabling tags to the login page, setting the associated security policy to require a token asserting claims of the types in which the user credentials are stored.

   (e) It adds a function to the login page to intercept the RSTR token that will later be returned by the CIdS.

   (f) It causes the PassCard logo to appear above the submit button, in such a way that clicking it invokes the CIdS.

4. The user clicks on the PassCard logo and the CIdS lights up.

5. The user selects and submits a PassCard. Alternatively, the user could create and choose a new PassCard. The CIdS creates and sends an RST to the SIP, which responds with an RSTR.

6. The CIdS passes the RSTR to the browser.

7. The browser extension performs the following tasks.

   (a) It intercepts and parses the SAML token (i.e. the RSTR).

    (b) If the token contains the URL of the target site, the extension compares it with the URL of the visited site, and only proceeds if they match.
    (c) It extracts the username and password from the pre-specified fields.
    (d) It auto-populates and auto-submits the login form.

8. The RP verifies the credentials and, if satisfied, grants access.

3.2.2 HTTPS-based PassCard

Steps 3d–8 below apply in the case where the RP uses HTTPS.

3. Following step 3c in section 3.2, the extension continues as follows.

    (d) It obtains the page's fully qualified domain name (FQDN), referred to below as the 'target URL'.
    (e) It causes the PassCard logo to appear above the submit field, in such a way that clicking it results in an HTTP redirect.

4. If the user clicks the logo, the browser is redirected to the pre-specified HS, and the target URL is also transmitted as a URL query parameter or as a hidden HTML form variable.

5. While interacting with the HS, the browser extension:

    (a) adds a 'hidden' password-login form to the returned HS page, if it does not already have one, at which point steps 3d–3f of section 3.2.1 are executed;
    (b) recovers and stores the target URL; and
    (c) transparently invokes the CIdS, at which point steps 5–7c of section 3.2.1 are executed.

7. The browser extension continues as follows.

    (d) It encrypts the username and password values with a secret key known only to the browser extension (see section 3.3.3).
    (e) It transparently redirects the user to the target URL, and the 'encrypted' username-password values are also transmitted as URL query parameters or as hidden HTML form variables.

8. While interacting with the target URL site, the browser extension:

(a) recovers and decrypts the username and password values;

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

(c) auto-populates and auto-submits the login form, after which step 8 of section 3.2.1 is executed.

3.3 Discussion

3.3.1 User Experience

The PassCard user experience when operating in HTTP mode is precisely the same as with 'conventional' password-based authentication except that, instead of manually entering and submitting a username and password, the PassCard user selects and submits a virtual card. The user experience in HTTPS mode is similar to that of HTTP mode, except that users (depending on their Internet speed, machine speed, etc.) may or may not experience a redirect from the target HTTPS site to the HS and vice versa, resulting in the temporary display of the HS web page. Note that, in both PassCard modes, users are not required to click the PassCard logo more than once or to remember any passwords.

3.3.2 CardSpace-enabled RPs

Regardless of whether or not an RP already supports CardSpace, the browser extension will always add the PassCard logo to the RP web page, as long as it detects username-password prompts on the page. This means that, if an RP supports CardSpace and simultaneously supports username-password authentication, as does the 'myOpenID' website[8] (sampled on 24/11/2010), the browser extension will still insert the PassCard logo above the submit button of the password-based login form. Informal tests on the prototype implementation suggest that this will not disrupt the normal operation of CardSpace.

The RP page will thus display both the CardSpace and the PassCard logos. In such a case, users will have (at least) three login options:

1. populating the username and password fields and submitting the login form manually;
2. using PassCard to auto-populate and auto-submit the login form; or
3. clicking the CardSpace logo to use CardSpace-based authentication.

[8] `https://www.myopenid.com/`

Note that, as Fig. 1 shows, hovering the mouse over the PassCard logo results in the display of text indicating that clicking it will activate PassCard.

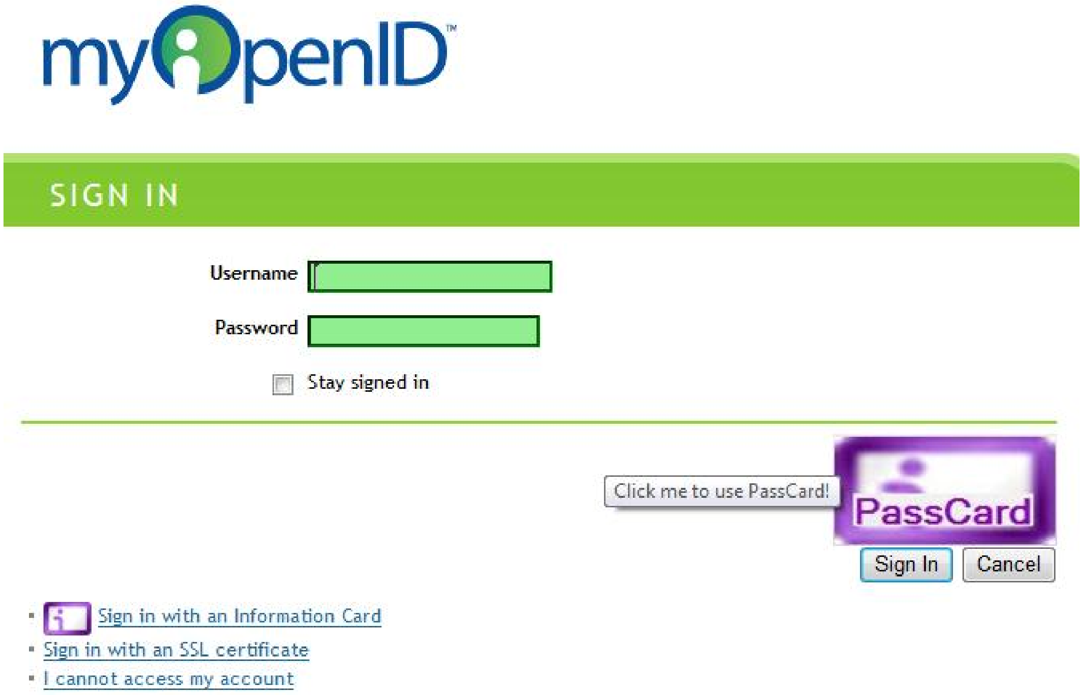

myOpenID™
SIGN IN
Username
Password
□ Stay signed in
Click me to use PassCard!
PassCard
Sign In
Cancel
Sign in with an Information Card
Sign in with an SSL certificate
I cannot access my account

Figure 1: PassCard co-operating with a CardSpace-enabled RP

3.3.3 Use of Cryptography

The encryption of the username and password in step 7d of section 3.2.2 is not necessary to prevent channel eavesdropping, because an SST/TLS channel is already established between the browser and the target HTTPS site. However, if the username and password are sent as part of the URL (as is the case in the PassCard prototype), then if sent in plaintext these values will be vulnerable to shoulder-surfing attacks since they will be shown in the browser address bar (and possibly also in the browser status bar). The prototype implementation uses a simple symmetric encryption scheme for username/password encryption to minimise the overhead.

4 Prototype Realisation

We next give details of a prototype implementation of the scheme. The prototype is coded as a plug-in using JavaScript [12], chosen because its wide adoption should simplify the task of porting the prototype to a range of other browsers. It uses the Document Object Model (DOM) to inspect and manipulate HTML pages and XML documents. The JavaScript code is executed using a C#-driven browser helper object (BHO), a Dynamic-link library (DLL) module designed as a plug-in for IE. Once installed, the BHO attaches itself to IE, thus gaining access to the current page's DOM. PassCard can readily be enabled or disabled using the add-on manager in the IE *Tools* menu. The PassCard prototype is an open source research project (`http://iescripts.org/view-scripts-808p1.htm` and/or `http://sourceforge.net/projects/passcard/`).

4.1 Registration

Prior to, or during, use of PassCard, the user invokes the CIdS and creates a PassCard, inserting their username in the *firstname* field and password in the *lastname* field. Optionally, the user could also insert the URL of the target website in the *web page* field[9]. For ease of identification, the user can give the PassCard a meaningful name, e.g. of the corresponding website. The user can also upload an image for the PassCard, e.g. containing the logo of the intended site. Example PassCards are shown in Fig. 2.

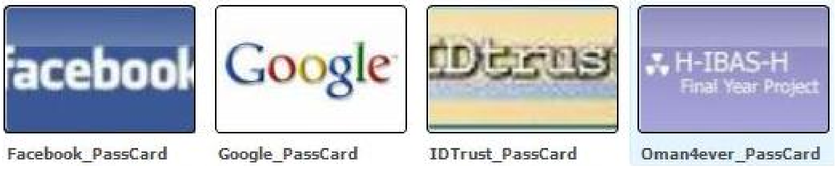


Figure 2: PassCards

4.2 Operation

The prototype implements the protocol steps specified in section 3.2. In step 3, the plug-in processes the RP web page in the following way.

[9] The *web page* field was chosen to contain the URL of the target website since it seemed the logical choice; however, this is an implementation option.

1. The plug-in 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) The plug-in 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 no input tag of type 'submit' is found, the plug-in 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 the plug-in highlights the username and password fields in green for ease of identification.

    The above process involves the following detailed processing.

    - Highlighting does not take place 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[10], etc.
    - When searching for the form submission button, if no submitting input tag is found then the plug-in searches for an 'image' tag. This is because, instead of a submit button, some websites display a clickable image[11] with similar functionality.

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

[11] 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.

- 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 (e.g. Google, YouTube, Yahoo, Gmail, SpringerLink) add a ‘Stay signed in’ or ‘Remember me’ check box between the password field and the submit button[12]. The plug-in addresses this issue by skipping all tags between the password field and the submit button, including those of type ‘checkbox’.

2. It uses the JavaScript inherent property ‘document.location.protocol’ to discover whether HTTP or HTTPS is in use.

4.2.1 HTTP-specific Implementation Details

In the following we refer throughout to the step numbers given in section 3.2.1.

- In step 3, the plug-in makes the following modifications to the login page provided by the RP.

  1. It adds an HTML object tag that allows the user to invoke the CIdS. Within the object tag, it sets the param tags to indicate that the RP security policy requires PassCards to contain two fields: the *firstname* and the *lastname* fields, (or three fields if protection against phishing is required, in which case the third field would be the *web page* field). Alternatively, the security policy could be configured so that the *web page* field is optional.
  2. It adds a function to the head section of the RP login page to intercept the XML-based RSTR message returned by the CIdS.
  3. It inserts the PassCard logo, causing it to appear just before the ‘login’ button in the login page, as illustrated in Fig. 3. The logo is associated with an ‘on-click’ event, so that, if clicked, the CIdS 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 PassCard logo, indicating that PassCard can be used to sign on.

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

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

1. It intercepts the XML-based RSTR message using the added function.
2. It parses the intercepted token and extracts the values of the *firstname* and *lastname* fields. 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.
3. It checks if the HTTPS-mode-signalling cookie (see section 4.3.1) is set; if so, it moves to step 7d of section 3.2.2. If not, it continues.
4. If a URL is present in the *web page* field of the RSTR message, it compares it with the URL of the visited website, and only proceeds if they match.
5. It automatically fills in the username and password fields with the *firstname* and *lastname* values, respectively.
6. It auto-submits the login form using the JavaScript ‘click()’ method.

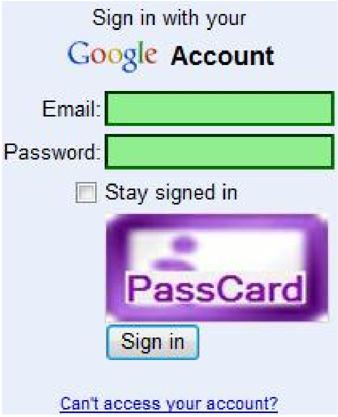


Figure 3: PassCard logo

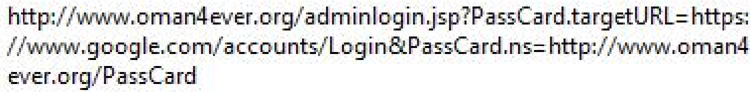
4.2.2 HTTPS-specific Implementation Details

In the following we refer throughout to the step numbers given in section 3.2.2.

- In steps 3d and 3e, the plug-in processes the RP web page in the following way.

    1. It obtains the HTTPS site's (full) URL, referred to as the 'target URL', using the JavaScript property 'document.location.href', removing any query parameters that may be attached to the URL.
    2. It creates an HTML hyperlink (a) tag, and sets its 'href' attribute to point to the HS. It also embeds an HTML 'img' (image) tag, that points to the PassCard logo, inside the hyperlink tag. In addition, it creates a title and alternative text, so that if the mouse is held over the PassCard logo, or if the image is not displayed, text is displayed indicating that PassCard can be used to sign on.

- In step 4, if the PassCard logo is clicked, the plug-in redirects the user to the HS. The target URL and the PassCard namespace are sent as query parameters (see Fig. 4).



http://www.oman4ever.org/adminlogin.jsp?PassCard.targetURL=https://www.google.com/accounts/Login&PassCard.ns=http://www.oman4ever.org/PassCard

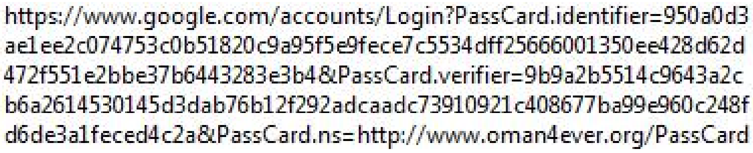
Figure 4: Redirect URL (target URL → HS)

- In step 5, the plug-in processes the HS web page in the following way.

    1. If no password-based login form is served by the HS web page, it creates a 'div' section and inserts an HTML form. The form includes a username (input tag of type 'text'), password (input tag of type 'password') and submit (input tag of type 'submit') elements. The 'div' section is then embedded in the HS page. To keep the changes to the HS page transparent to the user, the plug-in hides this form by setting the style of the 'div' section (which contains the added form) to 'visibility: hidden'. Note that embedding a form in the HS page enables the re-use of the code that was previously developed to handle HTTP mode.
    2. It parses the URL query parameters to obtain the target URL, and stores it in a cookie.
    3. It creates/sets the HTTPS-mode-signalling cookie (see section 4.3.1).
    4. It auto-invokes the CIdS using the JavaScript 'click()' method.

- In steps 7d and 7e, the plug-in:

  1. encrypts the username and password values using AES in CBC mode with a secret key known only to the plug-in.
  2. transparently redirects the user to the target URL, using the JavaScript property ‘window.location’. The encrypted username and password values and the PassCard namespace are sent as query parameters (see Fig. 5).



```
https://www.google.com/accounts/Login?PassCard.identifier=950a0d3
ae1ee2c074753c0b51820c9a95f5e9fece7c5534dff25666001350ee428d62d
472f551e2bbe37b6443283e3b4&PassCard.verifier=9b9a2b5514c9643a2c
b6a2614530145d3dab76b12f292adcaadc73910921c408677ba99e960c248f
d6de3a1feced4c2a&PassCard.ns=http://www.oman4ever.org/PassCard
```

Figure 5: Redirect URL (HS → target URL)

- In step 8, the plug-in processes the target URL web page in the following way.

  1. It parses the URL query parameters to obtain the encrypted values of the username and password, and decrypts them using its internally stored secret key.
  2. It locates the username, password, and submit fields.
  3. It auto-populates the username and password fields with the decrypted username and password values.
  4. It creates a three-second-long-living cookie to ensure that it does not re-submit the username-password values again before a three-second delay has elapsed (see section 4.3.3).
  5. It auto-submits the login form using the JavaScript click method.

4.3 Discussion

We now consider development issues of the scheme.

4.3.1 HTTP/HTTPS Modes

As a result of using the same program code to handle both the HTTP and HTTPS cases (to keep development efforts to a minimum and to maximise code efficiency), a cookie is used to signal to the browser extension that HTTPS mode is activated. This tells the browser extension to perform step 7d of section 3.2.2 instead of step 7d of section 3.2.1. More specifically, the HTTPS-mode-signalling cookie is created between steps 5b and 5c of section 3.2.2, and this cookie is checked between steps 7c and 7d of section 3.2.1; if set, HTTPS mode is activated, otherwise HTTP mode continues.

4.3.2 HTTP Server (HS)

The role of the HS in the PassCard system is not fully explained above. The HS can be any HTTP site, and is not actively involved in the protocol except to serve a web page when requested. It simply acts as a convenient way to avoid the problems associated with use of CardSpace with an HTTPS site. The choice of HS is not particularly privacy-sensitive in that it does not learn any sensitive data; it simply learns that a client at a particular IP address is using PassCard. The choice of HS is a configuration option in the prototype, so if the default site is regarded as privacy-threatening then a user can change it to any trusted address, e.g. that of a personal web page. In addition, users can use a local web server running on their own machines. Setting up a local web server is typically straightforward, e.g. by installing XAMPP[13].

The use of a personal web page also leads to other advantages, including minimising the load on the PassCard default HS. Also, if the default HS is down for whatever reason (e.g. hosting costs, network failure, etc.) users can always rely on or switch to their own HS. Moreover, the use of a personal web page is likely to make users feel more confident in its use, thereby encouraging the adoption of PassCard and CardSpace.

The HS could also be implemented as a proxy server. This would maximise user transparency in the sense that users would not experience a redirect from the target HTTPS site to another HS and vice versa. However, such a change would deny users the opportunity to use their own site. In addition, the proxy site will need to be trusted by the user, since username-password values will need to be given to the proxy server so that it can deliver them transparently to the target HTTPS site. Furthermore, if an HTTP proxy [10] is used, the user must force the browser to use the proxy. This is potentially inconvenient and may not always be possible (e.g. the user may not have permission to make the necessary change to the browser settings) [6]. Finally, PassCard would stop working if the proxy site was unavailable for any reason, e.g. networking failures, overloading, etc.

[13] http://www.apachefriends.org/en/xampp.html

4.3.3 Failed Authentication Attempts

When some websites, e.g. those of Microsoft Research[14] and RHUL's network authentication page[15], have incorrect credentials submitted to them, they do not change the URL (e.g. by redirecting the user to a new web page). Instead they inform the user of the failure of their authentication attempt on the same login page, maintaining the same URL in the browser's address bar.

When the PassCard plug-in is executing in HTTPS mode, such an event will cause it to run again, because the page has been refreshed but the URL is unchanged. When it runs, the plug-in will see the same URL with the same query parameters (as were redirected/sent by the PassCard plug-in when running on the HS's page), and will thus attempt to automatically sign in the user again with the same (wrong) credentials, because it believes the user has just been redirected from the HS. The website will deny access once again, and the same process will repeat indefinitely, resulting in either the user getting locked out after a certain number of failed authentication attempts or the browser becoming stuck in an (infinite) loop.

To address this issue, the plug-in creates a short-lived cookie just before attempting to auto-sign in the user for the first time, to ensure that it does not attempt to log in the user again before a certain period of time (e.g. 3 seconds) has elapsed. If the plug-in discovers that the user has been denied access following an attempt to automatically sign in, it informs the user and terminates. It also attempts to close the visited page, after first obtaining permission from the user. This termination is based on the assumption that the user has either inserted the wrong credentials in the current PassCard or has simply selected the wrong PassCard.

Other solutions to address this particular issue have been tried, but found not to work. For example, it is tempting to try to use the HTTP referrer[16] field, which can identify the web page that the user was visiting before they arrived at the current page. Indeed, JavaScript provides its own built-in property for HTTP referrer, namely 'document.referrer'. Thus, the PassCard plug-in could use this to attempt to sign in the user only if redirected from the HS; if not, it simply terminates on the assumption that the user has already attempted to sign in but has been denied access. However, informal tests suggest that this technique does not work for the PassCard plug-in, probably because the redirect from the HS to the target

[14] `https://cmt.research.microsoft.com/IDTRUST2010/Default.aspx`

[15] `https://nac.rhul.ac.uk/authentication/`, most recently checked on 24/11/2010.

[16] `http://tools.ietf.org/html/rfc2616#section-14.36`

site is initiated by the plug-in itself (using the 'windlow.location' property) and not by the HS server.

In any case, the use of the HTTP referrer field should be avoided because:

- if the HS and the target URL fall under the same domain, the HTTP referrer field would prevent the plug-in from automatically signing in the user; and
- the referrer field could be disabled or faked.

4.3.4 CardSpace-enabled RPs

We next discuss two alternative approaches to that presented in section 3.3.2.

1. After identifying that an RP supports username-password authentication, the browser extension could be configured to detect whether the RP already supports CardSpace; if so, the browser extension will deactivate PassCard. However, since the RP already offers passwords as a login option, offering PassCard may help password users and does not prevent users using the CardSpace-based authentication.
2. Instead of de-activating PassCard automatically if the RP supports CardSpace, as suggested above, the browser extension could ask users (e.g. via a JavaScript pop box) whether to activate PassCard. Although this would provide a greater degree of user control, repeated user prompting could become very intrusive. Nevertheless, this effect would be mitigated if the user's answer was stored.

5 PassCard Properties

We now consider a number of PassCard features and limitations.

5.1 Security

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

Furthermore, the CIdS identifies the RP to the user and indicates whether or not they have visited that particular RP before; if the user is visiting this

RP for the first time, CardSpace requests the user's permission to proceed[17]. This helps to support mutual authentication since the user and the RP are both identified to each other.

As with any local password manager, PassCard (when running in HTTP mode) 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.

PassCard reduces the threat of phishing attacks involving impersonation of legitimate sites by comparing the URL in the PassCard with that of the visited website. PassCard also supports the use of strong per-site passwords, since users no longer need to memorise or write down passwords.

Finally, note that the PassCard browser extension does not require any changes to default IE security settings, thereby avoiding potential vulnerabilities resulting from lowered browser security settings.

5.2 Usability

PassCard provides a simple, intuitive user experience through its use of the CIdS interface. At the same time, it familiarises users with CardSpace, thereby potentially facilitating future adoption of more secure means of authentication. Unlike other password managers which represent credentials in text form, PassCard credentials are stored in PassCards which can be equipped with a readily recognisable image, e.g. an RP logo.

PassCard operates completely transparently to external parties, and hence does not require any changes to RPs, identity selectors or to default browser security settings. The scheme is also highly flexible, since users can choose to use it simply by clicking the PassCard logo. It also supports flexibility in the choice of the HS when running in HTTPS mode.

Finally, by making use of CardSpace features, PassCard supports a degree of roaming. A user can transfer PassCards 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.

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

5.3 Limitations

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

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 PassCard 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.

The current PassCard prototype has not yet been tested with the recently released CardSpace 2.0[18]. We are thus unable to provide precise operational details for this version.

Like OpenID[19], we have used URL query parameters in the PassCard prototype to exchange data between the HS site and the target HTTPS site. This could give rise to the well-known issue of URL size limitations. For example, some 'older' servers may impose restrictions on the URL length; indeed the HTTP specification [5], whilst not specifying a maximum URL or path length, states that servers should be cautious about depending on URI (a generalisation of a URL) lengths greater than 255 bytes, because they may not be properly supported by some older client or proxy implementations. Additionally, the HTML 3 specification [13] states that the attribute value, e.g. a URL that was set as the value to the 'href' attribute in a hyperlink tag, is limited to 1024 characters[20]; however, the HTML 4 specifications [14] omit this restriction[21]. According to the Microsoft help and support centre, Microsoft IE supports a maximum URL length of 2083 characters[22], which is much larger than the length of a typical PassCard-generated URL (approximately 300 characters), as informal prototype testing suggests. Therefore, URL size limitations are not likely to be a major usability barrier.

[18] CardSpace 2.0 has not yet been standardised; in fact, at the time of writing, CardSpace 2.0 has only been released as a *Beta* prototype.

[19] http://openid.net

[20] http://www.w3.org/MarkUp/html3/HTMLandSGML.html

[21] http://www.w3.org/TR/html4/intro/sgmltut.html#h-3.2.2

[22] http://support.microsoft.com/kb/208427

Finally, older browsers (or browsers with scripting disabled) may not be able to run PassCard, 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.

6 Related Work

Password managers, which store passwords in a secure location either on the user PC or remotely, are now widely available. They typically store passwords in encrypted form and, unlike PassCard, require users to use a single master password to access the password store. Some are also capable of masking passwords, and others, much like PassCard, provide automatic password entry. Examples of password managers include open source schemes such as Password Safe[23], KeePass[24], Qubliette[25], Password Gorilla[26] and PINs[27] as well as commercial products such as RoboForm[28], Any Password[29] and Turbopasswords[30].

Perhaps the most distinctive feature of PassCard is its dependence on CardSpace, whereas other password managers are independent applications. PassCard can thus benefit from the CardSpace security features, which may give users greater confidence in its use. Most importantly, it is hoped that its introduction, with immediate practical benefits to the end user, will help encourage the adoption of more sophisticated identity management schemes like CardSpace. Such schemes offer the potential for a step forward in the practice of user authentication and authorisation, with potential benefits for all legitimate parties operating via the Internet. Indeed, without simple paths to adoption for schemes like CardSpace, there is a danger that it and all the other identity initiatives will fail.

7 Conclusions and Future Work

In this paper we have proposed a new version of PassCard, which (unlike its predecessor) supports almost all websites, including those using HTTPS.

[23] `http://passwordsafe.sourceforge.net/`

[24] `http://KeePass.info/`

[25] `http://tranglos.com/free/oubliette.html`

[26] `http://fpx.de/fp/Software/Gorilla/`

[27] `mirekw.com/winfreeware/pins.html`

[28] `http://roboform.com/`

[29] `http://anypassword.com/`

[30] `http://chapura.com/passwordmanager.php`

Users store their usernames and passwords in CardSpace personal cards, and use such cards to transparently sign on to corresponding websites. The scheme is based on a browser extension, and requires no changes to login servers; in particular, it does not require them to support CardSpace. Neither does the scheme require any changes to the current CardSpace identity selector, or to default browser security settings.

The scheme uses the CardSpace identity selector interface to seamlessly authenticate users to websites. It extends the use of personal cards to allow for transparent password management. Such an approach could help to extend the applicability of CardSpace.

Planned future work includes building a scheme that enables the use of CardSpace as a password-based single sign-on system. In addition, a possible topic for future work could be investigating the possibility of building a portable version of PassCard to support users who do not have installation privileges or are forced to use untrusted machines when travelling.

Acknowledgements

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

References

[1] 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 the IFIP IDMAN 2010 — 2nd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management*, volume 343 of *IFIP Advances in Information and Communication Technology*, pages 18–30. Springer, Boston, 2010.

[2] 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.

[3] Art Conklin, Glenn Dietrich, and Diane Walz. Password-based authentication: a system perspective. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS 04) — Track 7*. IEEE Computer Society, Los Alamitos, CA, 70170b, 2004.

[4] Marco De Luca. *Password Management for Distributed Environments.* VDM Verlag, Saarbrücken, 2008.

[5] R. Fielding et al. *Hypertext Transfer Protocol — HTTP/1.1.* RFC 2616, Internet Engineering Task Force, 1999. `http://www.ietf.org/rfc/rfc2616.txt`.

[6] Dinei Florêncio and Cormac Herley. One-time password access to any server without changing the server. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *Information Security, 11th International Conference, ISC 2008, Taipei, Taiwan, September 15-18, 2008. Proceedings*, volume 5222 of *Lecture Notes in Computer Science*, pages 401–420. Springer-Verlag, Berlin, Heidelberg, 2008.

[7] 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*, volume 5628 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 230–237, 2009.

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

[9] 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`.

[10] Ari Luotonen. *Web Proxy Servers.* Prentice Hall PTR, New Jersey, 1997.

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

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

[13] Dave Raggett. *HTML 3.2 Reference Specification.* W3C Recommendation, 1997. `http://www.w3.org/TR/REC-html32.html`.

[14] Dave Raggett, Arnaud Le Hors, and Ian Jacobs (editors). *HTML 4.01 Specification.* W3C Recommendation, 1999. `http://www.w3.org/TR/html401/`.

[15] Anita Sobe. *Single Sign-On in IMS-based IPTV Systems: Towards the Interworking of the Generic Bootstrapping Architecture and Liberty Alliance Identity Federation.* VDM Verlag and Co. KG, Saarbrücken, 2009.