A Browser-Based Approach to Smart Card Connectivity

Kapil Sachdeva, H. Karen Lu, and Ksheerabdhi Krishna
Gemalto, Inc., Technology & Innovation
9442 Capital of Texas Highway, Suite 400, Austin, TX 78759
{kapil.sachdeva, karen.lu, ksheerabdhi.krishna}@gemalto.com

ABSTRACT
Smart cards have provided security services for a wide range of applications including telecommunication, banking, and citizen identification. Connecting web applications with smart cards is a natural step forward to address some of the security issues in today's Web. The traditional approach for smart card based web applications provides security, but has the drawbacks of usability and flexibility. This paper presents a new browser-based approach utilizing web technologies. This approach focuses on communications and allows web applications to fully utilize smart card capabilities. The solution consists of two parts: a web browser extension that connects to the standard smart card communication layer; and a library that uses the browser extension and provides an API for web applications. This new approach departs from the traditional route, is much more natural for web applications, and presents several advantages.

Keywords: smart card, web application, security, web browser

1. Introduction

Security is a major concern for Web users as well as service providers. A promising way to enhance web security is to use smart cards. A smart card contains a secure and tamper-resistant microprocessor chip. It has secure memories, serves cryptographic functions, and is portable. Smart cards have provided security services for a wide range of domains. Extending smart cards to web applications is a natural step forward. To achieve this, smart cards must be able to connect to web applications.

A smart card connects and communicates with a personal computer or a cell phone to provide services such as secure storage of private keys, encryption, and digital signature. Personal computers implement an industry wide standard known as PC/SC for accessing smart cards and writing card reader drivers [1]. An implementation of the PC/SC standard is available in all major personal computer operating systems (OS). Operating systems also offer a device independent cryptographic API to insulate developers from different ways of obtaining cryptographic services. For smart cards this cryptographic API is aided by a card specific service provider implementation (Figure 1).

Web browsers can access smart cards through the aforementioned cryptographic API. However, the introduction of new web browsers and variations of cryptographic APIs across OSs give rise to a myriad set of browser, operating system, and cryptographic API combinations, which presents challenges, such as cross browser availability, development, distribution, and usability, for web applications using smart cards.

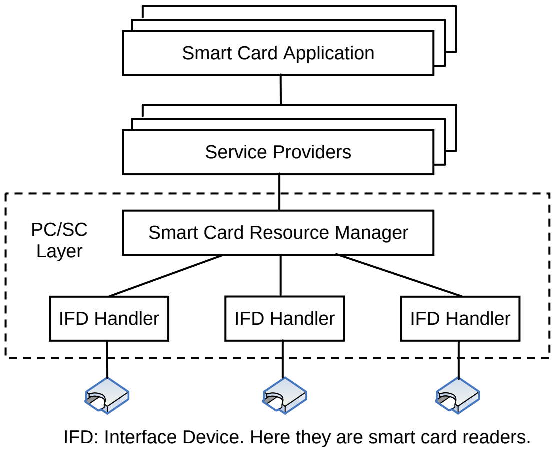
Smart Card Application
Service Providers
PC/SC Layer
Smart Card Resource Manager
IFD Handler
IFD Handler
IFD Handler
IFD: Interface Device. Here they are smart card readers.

Figure 1. PC/SC architecture

These challenges necessitate a new approach for web applications to use smart cards. This paper proposes a new browser-based approach that utilizes web technologies. This new approach focuses on communications and allows web applications to fully utilize smart card capabilities. The solution consists of two parts. The first part is a web browser extension that connects to the PC/SC layer and hence enables the browser to communicate with a smart card. The second part is a library that uses the browser extension and provides an API for web applications. With this setup, web applications can use the library to access smart cards. This solution departs from the traditional approach and is much more natural for web applications.

Our approach addresses major issues faced by smart card access via browsers and has several advantages. First, web applications can fully utilize smart card capabilities without being limited to the OS specific cryptographic API. Secondly, smart card specific modules are delivered on-demand from the server, eliminating the complexity of software update on PCs. Thirdly, our browser independent library enables web applications to offer a consistent user experience. Finally, our approach enables a trusted web server and a smart card to establish a secure channel with no operating system specific intermediaries. We call the implementation of our approach *SConnect*.

2. Traditional Approach

Applications traditionally access smart card functionality through a device independent cryptographic service API provided by the host operating system. Microsoft's Cryptographic API (CAPI) [2] natively integrated in Windows, Apple's CSSM API of the Common Data Security Architecture (CDSA) implementation natively integrated in OS X [3], and the PKCS#11 (Public-Key Cryptography Standards) API implementing RSA Laboratories' PKCS#11 Specification [4] available across major operating systems are the three cryptographic service APIs in use today. Although the PKCS#11 API is considered interoperable across operating systems, native integration of OS specific cryptographic APIs offer advantages such as updates, better application integration, and platform consistent usage experiences.

While cryptographic APIs offers benefits they also pose drawbacks:

1. *Functionality.* The cryptographic API defines a specific set of functionalities, mostly supporting Public Key Infrastructure (PKI) [5]. Other smart card services or emerging standards must either be accommodated via non-standard means or wait for updates from OS vendors.

2. *Card modules.* Due to differences in smart cards, card manufactures (or third parties) write and deploy card specific modules. These modules must be installed on users' computers. Installation and updates are inconvenient for end users.

3. *Application Programming Interface.* CAPI, PKCS#11, and CDSA have different non-interoperable APIs. Web developers have to write browser-dependent code in order to use the appropriate API or architect a complex abstraction of their own.

4. *User experience.* User interfaces for various devices and cryptographic operations differ across operating systems, which present different user experiences across browsers and operating systems yielding confusion.

5. *Secret keys.* Off-card applications establishing a secure communication channel with smart cards use embedded keys making them vulnerable to decompilation.

These issues have hindered widespread deployment of smart card solutions for web applications (and in some cases for client applications).

3. A Browser-based Approach

A web application typically consists of two major parts, one executes on a server and the other in a browser. The server part of the application implements server side business logic, interacts with backend systems, and generates dynamic HTML content to serve the client. The browser part of the application renders content, implements client side logic, interacts with the user, and executes scripts, typically JavaScript.

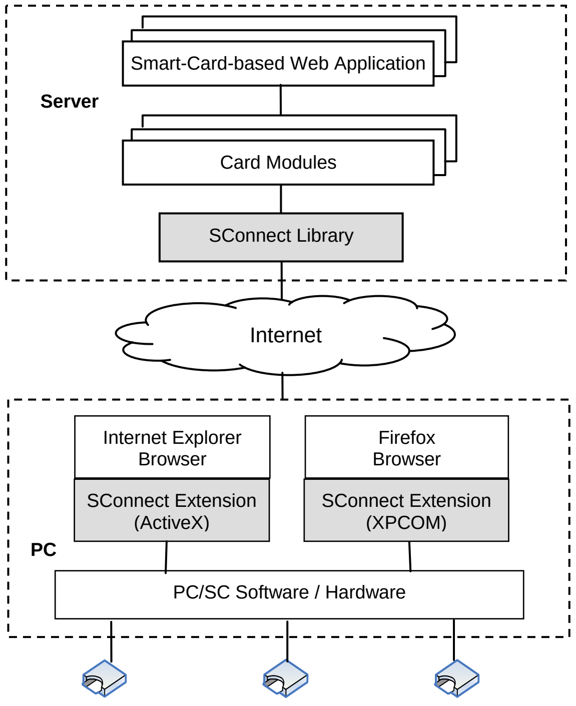
Server
Smart-Card-based Web Application
Card Modules
SConnect Library
Internet
Internet Explorer Browser
SConnect Extension (ActiveX)
Firefox Browser
SConnect Extension (XPCOM)
PC
PC/SC Software / Hardware

Figure 2. SConnect-based web application architecture.

A smart card connects to a user's computer. To access smart card functionality, a web application must communicate with the smart card. SConnect enables such communication. It consists of two parts, a web browser extension and a JavaScript library. The browser extension extends the PC/SC layer, which enables programs or scripts in the web browser to communicate with smart cards. The SConnect library provides a JavaScript API for developers to write web applications that connect to and access smart cards. The library uses the browser extension to communicate with smart cards. Figure 2 illustrates the architecture of a SConnect-based web application. Typically client side JavaScript code in the web application resides on a web server and runs in a web browser on demand.

Conceptually SConnect behaves similar to the XMLHttpRequest object. While XMLHttpRequest provides connectivity between JavaScript and the server, SConnect provides connectivity between JavaScript and the smart card. The browser extension provides a wrapper to the PC/SC stack and enables JavaScript running in a browser to access smart cards through PC/SC (Figure 3).

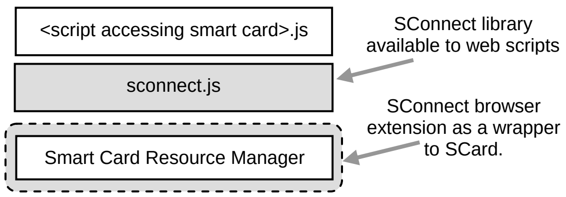
<script accessing smart card>.js
sconnect.js
Smart Card Resource Manager
SConnect library available to web scripts
SConnect browser extension as a wrapper to SCard.

Figure 3. Components of SConnect.

To ease development, SConnect hides browser dependent complexities from web application developers. The SConnect library provides utility functions that handle extension detection, installation, and update. The SConnect extension is less than 500KB and is currently available for IE and Firefox (Windows, OS X and Linux). It will eventually be available for all major browsers across all PC platforms.

Accessing the smart card environment over the Web poses security risks and challenges. To mitigate these risks, SConnect deploys a set of security mechanisms to protect end users. These measures include digital signature of the browser extension, enforcing HTTPS, user consent, server validation, and an authorization mechanism called *Connection Key*.

Digital Signature: The browser extension is digitally signed using a code signing key issued by VeriSign. A signed extension instills confidence in users by validating the source of the extension.

Enforce HTTPS: To ensure secure communications with the remote web server and to prevent Man-in-the-middle attacks (MITM), SConnect requires HTTPS connections between a browser and a remote web server for the web application to access the smart card.

User Consent: The first time a user visits a SConnect-enabled website, SConnect presents a warning dialog informing that the website is trying to access the smart card. The user must examine the warning and make a conscious decision to allow a website to access the smart card. The user can allow or deny the access, and have SConnect remember his decision for future visits.

Server Validation: During SSL handshaking, the browser receives and examines the website's SSL certificate. When the browser determines that certificate is invalid, it presents a warning to the user. Many users ignore the warning and continue, which may land the user at a malicious website or fall victim to MITM.

To mitigate this risk, SConnect does additional server SSL certificate validation when a web application tries to access a smart card. The validation includes verifying the signatures of the certificate chain, ensuring that the root CA is trusted by the browser, checking the validity period, and matching the Common Name in the certificate with the URL of the website. If SConnect determines that the certificate is invalid, it will not allow any connection between the website and the smart card, even if the user has accepted the browser connection. SConnect considers self-issued SSL certificates invalid. Server validation prevents malicious websites or MITM attackers to access the user's smart card.

Connection Key: While HTTPS ensures a certain level of protection to users, companies issuing SSL certificates do not always employ consistent issuance practices. To add another layer of protection, SConnect employs Connection Key, which uniquely binds the website deploying the SConnect-based application with the web browser extension. The issuer (Gemalto) of the browser extension uses its private key K_{priv} to sign Connection Key. The corresponding issuer public key, K_{pub}, is encoded within the browser extension. The owner of a website must go through a strict vetting process in order to obtain a SConnect Connection Key.

The Connection Key does not contain any secret. It includes a set of attributes namely, Common Name (the website domain name); issuer name; issue date; expiration date; and digital signature signed by the issuer.

When establishing a SConnect session, a website must present its Connection Key. SConnect validates the key by checking the validity of its attributes: the Common Name must match the website's domain name; the expiration date must be at or beyond the current date; and the signature must pass verification using the issuer public key K_{pub}. A connection is allowed only if the validation of the Connection Key is successful. Otherwise, the connection request is denied.

4. Analysis and Related Work

We now see how the SConnect-based approach compares with the traditional (cryptographic API) approach in addressing the drawbacks listed in Section 2.

1. *Functionality.* SConnect-based web applications can fully utilize smart card capabilities instead of being limited to PKI functions, because the SConnect extension is a wrapper over the PC/SC layer.

2. *Card modules.* SConnect-based web applications move card specific modules from individual PCs to the server. The modules are delivered on demand. Applications are updated at the server eliminating the complexity of pushing updates to individual PCs. The small footprint of the browser extension makes it quick

and easy to install and update using browser-based methods.

3. *Application Programming Interface.* The SConnect library presents a browser-independent JavaScript API. Web application developers can have the same smart card access code work across browsers.

4. *User experience.* SConnect's browser independent application interface enables building web applications that provide a consistent user experience across browsers and platforms.

5. *Secret keys.* SConnect-based web applications manipulate secret keys only on the server. This limits key exposure and access.

Another approach to provide smart card services for web applications is to use the Java Applet [6]. This approach suffers from the drawback of requiring an install of a Java Runtime Environment (JRE), and the burden of having to deal with the nuances of cross-platform Java development. The JRE is not lightweight and not preferred in some environments.

SConnect enforcing HTTPS and a valid server certificate is similar to previous work on protecting high-security websites [7]. SConnect extends this to controlling access to smart cards.

The Connection Key adds another layer of defense. For example, a Phishing website www.go0d.com, which has a valid SSL certificate, may mislead the user who intends to go to www.good.com. But since www.go0d.com does not have a valid Connection Key the connection from www.go0d.com to a smart card is rejected. While SConnect cannot defend against all attacks [8], it can resist some classical MITM, Phishing, and Pharming attacks by demanding HTTPS and validity of server certificate.

SConnect enables a secure connection from a legitimate website to a smart card and contains mechanisms to prevent a malicious website from accessing the smart card. However, SConnect does not address browser vulnerabilities or provide protection against malware.

SConnect and its built in security mechanisms do not absolve web applications from following web development security best practices [9]. For example, to prevent imported libraries, such as sconnect.js, and scripts from being replaced by attackers, all resources imported or used by the web page should use HTTPS explicitly or obtain them via relative path [10].

5. Conclusion

The new approach for smart card connectivity presented in this paper extends PC/SC to web applications and bridges the gap between smart cards and the Web. It shifts the complexities of card dependencies, services and updates from the PC to the server. It is easier to update, and more user friendly than the traditional approach. The resulting application architecture can be used in contexts ranging from traditional smart card use to card administration, management and applications in emerging identity management frameworks. Readers can find more information from http://www.sconnect.com.

6. Acknowledgements

The authors gratefully acknowledge the feedback received from Guillaume Huysmans, John-Philip Wilson and the referees.

7. References

[1] PC/SC Workgroup, *Specifications*, http://www.pcscworkgroup.com, V 2.01.3, Jan. 2006.

[2] Microsoft, *Cryptographic Service Providers*, http://msdn.microsoft.com/en-us/library/ms953432.aspx

[3] Apple, *Mac OS X Security Framework*,

[4] RSA Laboratories, *PKCS#11: Cryptographic Token Interface Standard*, http://www.rsa.com/rsalabs/node.asp?id=2133.

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

[6] Svetlin Nakov, *Java Applet for Signing with a Smart Card*, developer.com, https://www.developer.com/java/other/article.php/3587361, Apr. 2006.

[7] C. Jackson and A. Barth, *ForceHTTPS: Protecting High-Security Web Sites from Network Attacks*, Proceedings of WWW 2008, Apr. 2008, Beijing, China.

[8] C. Karlof, J.D. Tygar, D. Wagner, and U. Shankar, *Dynamic Pharming Attacks and Locked Same-origin Policies for Web Browsers*, 14th ACM Conference on Computer and Communications Security, Oct. 2007, Alexandria, Virginia, USA.

[9] Open Web Application Security Project (OWASP), http://www.owasp.org/index.php/Main_Page.

[10] C. Jackson and A. Barth, *Beware of Finer-Grained Origins*, Proceedings of Web 2.0 Security and Privacy, May 2008.