A Pragmatic Online Authentication Framework using Smart Cards

H. Karen Lu, Asad Ali, Kapil Sachdeva[1]
Gemalto
Austin, Texas, USA
{Karen.lu, asad.ali}@gemalto.com

Ksheerabdhi Krishna
Gemalto
La Ciotat, France
Ksheerabdhi.krishna@gemalto.com

***Abstract* - Like most security systems, designing a secure two-factor online authentication framework is hard, but designing one that is also intuitive to use and easy to deploy is even harder. While a secure, but overly complex framework may offer little security in the end since it never gets used, an overly simplistic one that focuses merely on usability may gain initial acceptance but will inevitably lead to data breaches. To address this design paradox, we present a new online authentication framework that provides security, usability, and ease of deployment. This framework combines the proven hardware security of smart cards and the universal ease of web access through browsers, without imposing the deployment and usability complexities generally associated with conventional smart card systems. The resulting authentication solution is applicable to existing smart cards already deployed, intuitive for users, and convenient for service provides to both develop and maintain.**

Keywords-Authentication; security; smart cards; usability.

I. INTRODUCTION

Internet has undoubtedly been a phenomenal success, dominating every facet of our professional and social life. However, this success has partly come at the expense of a continuous barrage of security attacks against both users and service providers. Attackers employ various mechanisms to steal user's credentials. Some use social engineering to lure naïve users into revealing their credentials [1], while others leverage network security flaws and web application vulnerabilities to attack web servers and their databases [2]. These attacks compromise confidential user data. Some of this data can actually be user authentication credentials that enable attackers to impersonate users and gain subsequent access to additional user data and services. This is generally referred to as identity theft. Such theft is possible partially because a vast majority of online service providers still rely on username and password, a weak single-factor authentication method. Furthermore, since users tend to use the same password on multiple service providers [3], it amplifies the potential damage resulting from a stolen credential.

The weakness of password based authentication solutions can be addressed by using an authentication method that relies on multiple factors for verifying a user's identity. For example, in addition to password, the what-you-know factor, the authentication method may also require a what-you-have factor in the form of a separate physical token, or even a what-you-are factor in the form of biometric information. While there is some social skepticism around the use of biometric information, the use of dedicated physical tokens to provide a second authentication factor that compliments passwords is gradually gaining acceptance with service providers dealing with high value transactions [4]. In general however, we still see a lot of not-so-secure systems in use. One reason for this could be the inertia of status quo; it is always hard to change an existing framework. Another reason is what we call economies of convenience. This notion is somewhat analogous to the economies of scale, a microeconomic term that refers to the cost advantages that a business obtains due to expansion. Similarly, there is also a cost advantage to having systems that are extremely convenient to use, even if they are not as secure. Enterprises can then develop risk models of dealing with data breaches, when they happen. As for the average end-users, they generally turn a blind eye to security vulnerabilities as long as the systems they use are convenient, and security threats not imminent.

However, a continued increase in the intensity and frequency of cyber attacks is beginning to challenge these well established economies of convenience. Enterprises will eventually mandate stronger security measures once it makes better economic sense for them to lower the cumulative cost of data breaches by reducing the risk instead of managing this risk with their current models. We can reach this watershed moment either through an exponential increase in the number of data breaches, or by designing security systems that are more convenient to develop, deploy, use, and manage. It is the intent of this paper to propose a solution for the later.

The rest of the paper is organized as follows. Section II describes why smart cards are excellent candidates for use as authentication tokens. Section III describes the existing smart card infrastructure and explains how it hinders wide spread adoption of smart cards. Section IV introduces SConnect technology that addresses the issues identified in Section III. Section V describes a two-factor online authentication solution based on SConnect and

1. This work was completed while Mr. Sachdeva was with Gemalto. Mr. Sachdeva is now working with HID.

 ISBN: 978-1-61208-152-6

Section VI offers security and usability analysis of this solution. We conclude with Section VII.

II. AUTHENTICATION TOKENS

Physical tokens for multi-factor online authentication generally use one of the two common authentication techniques; One-Time-Password (OTP), or X.509 certificate based challenge and response. In both cases the hardware processor of the token uses private keys to perform cryptographic computations for generating a "credential" that is unique for each authentication attempt, and therefore can neither be stolen from the web server, nor replayed by an attacker. Since the token stores the private cryptographic keys, the strength of such an authentication method is a function of the token's hardware security.

Smart cards are excellent candidates for these physical tokens. They are tamper resistant, portable, and secure microprocessor devices that have been widely used in a variety of applications related to both physical and logical security. The smart card does not usually have its own power supply, yet it operates as a very small computer with an embedded operating system (OS) that controls application execution, access restrictions and communication with the outside world. However, unlike the mainstream personal computers, smart cards offer much greater hardware security. It is extremely difficult to compromise data stored inside the smart cards. This is because smart cards are designed with a heavy focus on security from the ground up, and this focus is maintained throughout their lifecycle. As such, smart cards can withstand attacks based on physical probing, logical probing, side channel threats, fault induction and software debugger probing [5]. A more detailed discussion of techniques for preventing such attacks is outside the scope of this paper.

Suffice to say that smart cards can serve as excellent tokens of two-factor authentication. However, despite their hardware advantage, smart cards are yet to garner widespread adoption outside their controlled niche markets. One reason for this lackluster acceptance is the complexity of deploying smart card based solutions, and the inconvenience of using them. To address these problems this paper introduces a new two-factor online authentication framework. It supports an X.509 certificate-based challenge-response model of authentication using smart cards, and utilizes a unique communication model that allows seamless access to smart card functionality directly from web applications. This approach facilitates easy adoption by end users as well as service providers.

III. CURRENT SMART CARD FRAMEWORK

In order to appreciate the value of the new method, we first have to consider how smart cards are currently used for online authentication. This use is somewhat restricted to environments where it is viable to create and maintain smart card specific infrastructure. To use smart card services, host applications must be able to communicate with smart cards. This communication component has been the critical piece of all authentication systems based on smart cards, and is perhaps the reason why smart cards have thus far not enjoyed widespread adoption in security frameworks for ubiquitous and loosely managed systems. In this section, we describe the conventional smart card connectivity model with respect to the X.509 based online authentication, and the usability and deployment issues inherent in the existing methods.

A. Smart Card Middleware

Conventional smart cards use traditional ISO 7816 communication protocols to talk to their host devices. These devices range from mobile phone handsets to custom readers at public transportation terminals. In such environments smart cards continue to be useful and well integrated components. Conventional smart cards are also used in online authentication applications, though their acceptance in this market has been less successful. Two key reasons for this are the lack of built-in smart card reader drivers on mainstream PCs, and the need of smart card specific middleware; both of which are barriers for entry into the online market that demands ubiquitous plug-n-play behavior. Although the reader driver issue is addressed in modern computer operating systems (OSs) through the standard USB CCID class driver for smart card connectivity, the distribution of smart card middleware continues to impede the adoption of online authentication solutions.

This middleware enables application programs to access the cryptographic functionalities of smart cards (or other security devices) without worrying about the details of these devices. For this purpose, PC OSs offer a device independent cryptographic API, which is realized by device specific implementations. Different operating systems have their own APIs, and different devices (including smart cards) require their own implementations. Middleware examples include Microsoft's CyptoAPI, RSA Laboratories' PKCS#11, and Apple Computer's CDSA. While offering similar capabilities, they present different APIs and have additional restrictions that may limit the functionalities of applications developed for a specific middleware API. For example, CryptoAPI is used within the native Windows ecosystem [6] but not supported on Mac, Linux, or even browsers other than IE on Windows. The PKCS#11 specification [7], though available across all major operating systems, is natively accessible only via Firefox browser, and is not supported by IE. Similarly, CDSA is only supported on Mac OS X [8].

 ISBN: 978-1-61208-152-6

To further complicate matters, the user may need to manually install these browser/OS specific middleware on all the machines he intends to use. For example, to use Firefox browser with a smart card, the user needs to download and install the PKCS#11 library. Such manual installations severely restrict the portability of smart cards. While the two-factor authentication credentials are stored in a device which you can carry in your wallet, the use of these credentials is not portable. Furthermore, since middleware of a particular smart card may not be available for all browser/OS combinations, it restricts the online authentication solution to a limited number of platforms.

B. X.509 Authentication

The X.509 certificate based authentication utilizes a user's digital certificate along with the corresponding private key. Using public key cryptography [9] the user can demonstrate that he is indeed the holder of the private key. This can be done by a user storing his private key and using it in calculating response to a server challenge when needed. Unlike passwords which a user can remember, certificates and private keys are blobs of data that need to be securely stored in digital form. To ensure flexibility and inter-operability, the security industry has specified architectures for the storage and use of these credentials either from the operating system of host computer or from an external security device such as a smart card. Since hardware tokens and even software security devices present different interfaces and use different protocols, these architecture specifications provide a common bridge for accessing the cryptographic capabilities of these devices. For example, certificate access, document signature and encryption, and card holder validation can now be done in a device neutral way from a given platform. The middleware mentioned earlier implements some of these specifications. A web browser can use this middleware to accomplish SSL/TLS mutual authentication with the client certificate and private key stored in a smart card. However, the smart card middleware is a local resource; web applications cannot use it to access smart cards in a platform neutral way.

C. Online Authentication Usability

Even if the hurdle of middleware installation is overcome, there can be usability issues. Smart card functionality is accessed via the cryptographic interfaces of web browsers. These interfaces are agnostic to the underlying credential store (smart card, host computer, etc.) and therefore, provide broad abstractions. However, abstractions by their very nature are written at a high level, and seldom address all the specificities of a target device. Because of this, security mechanisms based on smart card conventional connectivity are generally seen as road blocks to application efficiency and often abandoned. Furthermore, certain web browsers, such as IE, require the user to propagate certificates from his smart card to web browser's persistent certificate store. This makes the smart card based online authentication non-portable by limiting its use to only those computers to which such propagation has been done.

Let us look at another usability aspect by considering the following example:

1. A user browses to a website that requires certificate based authentication.
2. The web browser displays a list of certificates propagated from the user's smart card.
3. Since each certificate has a specific use, the user is asked to select the appropriate certificate.
4. Once the user selects the certificate he is prompted for a PIN.
5. The user enters the PIN and authenticates successfully.

While this appears simple, Steps 2, 4, and 5 present a user interface challenge. They present the user with a UI that is specific to the browser, host operating system and the smart card middleware. The web application has no control over the way the user interacts with these UI elements. For example, the tasks of canceling the certificate selection, requesting smart card insertion, physically removing the smart card, or abandoning the PIN entry, could provide inconsistent responses. Furthermore, the experience of accessing the same web site varies with each web browser and operating system.

The current smart card connectivity model is therefore not a panacea for achieving a seamless marriage between security and usability. While the notion of carrying your credentials in a secure portable device is a fascinating idea, it fails to germinate into a viable solution that utilizes these credentials for online authentication. We address these issues through a new smart card connectivity method called SConnect, and then show how it can be used to design a smart-card-based online authentication framework.

IV. SCONNECT

SConnect [10] is a connectivity bridge between a smart card and a web application. A web application typically consists of two major components: a server part that executes on a remote web server; and a client part that executes in the local web 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 client part of the application renders web content, implements client side logic, interacts with the user, and executes scripts, typically JavaScript. To access the functionality of a smart card connected to a host computer, a web application must communicate with the smart card. SConnect enables this communication, without requiring

 ISBN: 978-1-61208-152-6

the installation of any conventional smart card middleware.

A. SConnect Architecture

The SConnect architecture is composed of two parts: a web browser extension; and a library. The web browser extension extends the standard computer and smart card interface layer (called PC/SC) to enable client applications written in JavaScript to communicate with the smart card. 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 1 illustrates the architecture of a SConnect-based web application, with the two shaded boxes representing the two parts of SConnect. Typically the client side JavaScript code in the web application resides on a web server and is downloaded to run in the web browser on demand. Some common code, which interacts with smart cards using the SConnect library, is referred to as smart card module, and is different for each type of smart card. In the conventional approach such differences between smart cards are handled by installing different middleware components, a process that is both difficult to maintain and cumbersome to use. By contrast SConnect allows such support by simply downloading a different JavaScript file, a process that is completely transparent to the user.

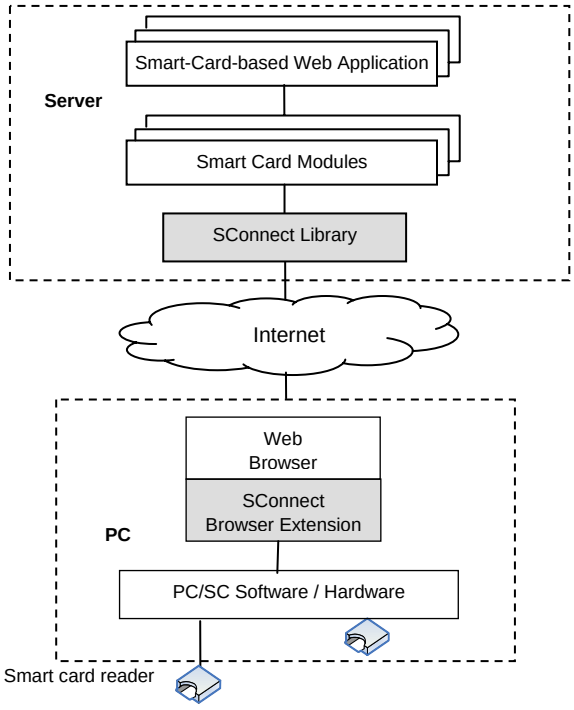

Figure 1. SConnect-based web application architecture.

To ease development, SConnect hides browser dependent complexities from web application developers. The SConnect library provides utility functions that handle the detection, installation, and update of SConnect browser extension. This extension is less than 500KB and is available for most common web browsers on Windows, OS X, and Linux operating systems.

B. SConnect Security Features

While the openness of SConnect that allows direct access to smart cards is a bonanza for web application development, it also broadens the attack surface. Malicious applications can potentially use the same interface to connect to the smart card and use its cryptographic services to impersonate the card holder. To mitigate such potential risks, SConnect deploys a set of security measures to protect the end user and service provider. These measures include digital signature of the browser extension, enforcement of HTTPS, user consent, server verification, and a control mechanism called Connection Key.

Digital Signature: The SConnect browser extension is digitally signed using a code signing key issued by a trusted certificate authority, such as VeriSign. A signed extension instills confidence in users by validating the source of the extension.

Enforcement of HTTPS: To ensure secure communication with a remote web server and to prevent Man-in-the-middle (MITM) attacks, SConnect mandates HTTPS connection between the browser and the remote web server before a web application is allowed to access the smart card. SConnect rejects connection requests from non-HTTPS connections.

User Consent: The first time a user visits a SConnect-enabled website, SConnect displays a warning message box informing the user that the website is trying to access the smart card. The user must make a conscious decision to allow or deny such access. SConnect can save this decision for future reference if so desired by user.

Server Verification: During SSL (or its predecessor TLS) handshaking, the browser receives and examines the server website's SSL certificate. If this certificate is invalid, the browser presents a warning to the user. However, most users ignore such warnings and continue anyway, thereby exposing themselves to malicious websites and MITM attacks. To mitigate this risk, SConnect does additional server SSL certificate verification when a web application tries to access the smart card. This verification consists of 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

 ISBN: 978-1-61208-152-6

between the website and the smart card, even if the user has accepted the browser connection.

Connection Key: While the server verification ensures the identity of a website, it does not make any claims about its trustworthiness. That determination has traditionally been left at the user's discretion - a task that is made even harder by the promiscuous approach to issuance of SSL certificates followed by some certificate authorities, even for the Extended Validation Certificates [11]. To address such risks, and also to introduce a licensing policy, SConnect employs the Connection Key. The authority that issues smart cards can decide which web portals can access these cards and, hence, can issue the Connection Key to only these portals. Examples of such smart card issuing authorities can be governments that issue smart cards to their citizens and want to control at which government service portals that citizens can use these cards.

The Connection Key uniquely binds to the SSL certificate of the website that deploys SConnect-based applications. This ensures that only websites with valid Connection Keys can access the smart card. The Connection Key itself does not contain any secret. It includes a set of attributes such as Common Name (the website domain name), issuer name, issue date, expiration date, and hash of the website's SSL certificate. This information is then signed using the SConnect extension issuer's private key, K_{priv}. The corresponding issuer public key, K_{pub}, is encoded within the SConnect browser extension. SConnect can therefore verify the Connection Key and ensure that the common name in the Connection Key matches the domain name the web browser is currently connected to.

These measures ensure a greater level of trust between the end user and service provider so that the openness of SConnect architecture can be utilized in online applications without reducing the security associated with conventional middleware approaches. The next section describes how this open, yet controlled access is used to design a secure two-factor online authentication framework.

V. TWO-FACTOR AUTHENTICATION

We propose a smart-card-based user authentication method for online access that does not rely on the conventional middleware for connecting to the smart card. Instead it uses SConnect. The authentication is based on a classical challenge-response protocol that uses X.509 certificate and the corresponding private key stored in the user's smart card. What makes this method unique is the benefit it brings to service providers and users alike. Web applications based on this authentication method are easy to develop, deploy, use and maintain.

The authentication software consists of two parts: a server part that resides and runs on the web server; a client part that is dynamically downloaded from the web server, but is executed in the web browser. The server component is responsible for authenticating the user, managing login sessions, logging events, and interacting with certificate authorities or issuers for verifying X.509 certificates. The client component renders the user interface in the web browser for user interaction. It also uses the SConnect extension to connect with the smart card and use its cryptographic services.

When authenticating a user to an online server, located on domain D, the authentication involves the following cryptographic operations:

1. The online authentication server with domain D generates a random challenge $C = \{r, D\}$, which is unique for each authentication request. This challenge is generated by combining a random sequence of bytes r, with the domain of the server, D. The server sends this challenge C to the smart card through the web browser and SConnect.
2. SConnect compares the domain D encoded in the challenge C with the current domain D_b that the browser is connected to. If $D = D_b$, SConnect forwards the challenge to the smart card. Otherwise, SConnect rejects the connection. The authentication fails.
3. If SConnect forwards the challenge C to the smart card, the card digitally signs the challenge using the private key K_{priv}. The resulting signature is the response R:
   $R = sign\{C\} = RSA_{Encrypt}\{SHA\text{-}1(C)\}K_{priv}$
4. The response R is sent back to the authentication server along with the X.509 user certificate read from the smart card. The server verifies the signature using the public key, K_{pub} , retrieved from the user's certificate. The computation as follows:
   $H_b = Decrypt\{R\}K_{pub}$
   $= RSA_{Decrypt}\{RSA_{Encrypt}\{SHA\text{-}1(C)\}K_{priv}\}K_{pub}$
   $H = SHA\text{-}1(C)$
5. *SHA-1( )* is a cryptographic hash and the chance of collision is therefore extremely small. When $H_b = H$, the authentication server concludes that the user's smart card does indeed hold the private key. The user is authenticated. Otherwise, the authentication fails.

Appending the current domain to the challenge in step 1 helps defend against the Man-In-The-Middle and the chosen protocol attacks. (See Section VI.) The authentication is accurate since it is based on cryptography and there is no uncertainty involved. Figure 2 illustrates the message flow of this authentication process.

The user logs in to a website through the authentication server. This connection is over HTTPS protocol, and the web browser performs server authentication as part of the SSL handshake process. The

 ISBN: 978-1-61208-152-6

rest of the numbered steps in user authentication are listed below.

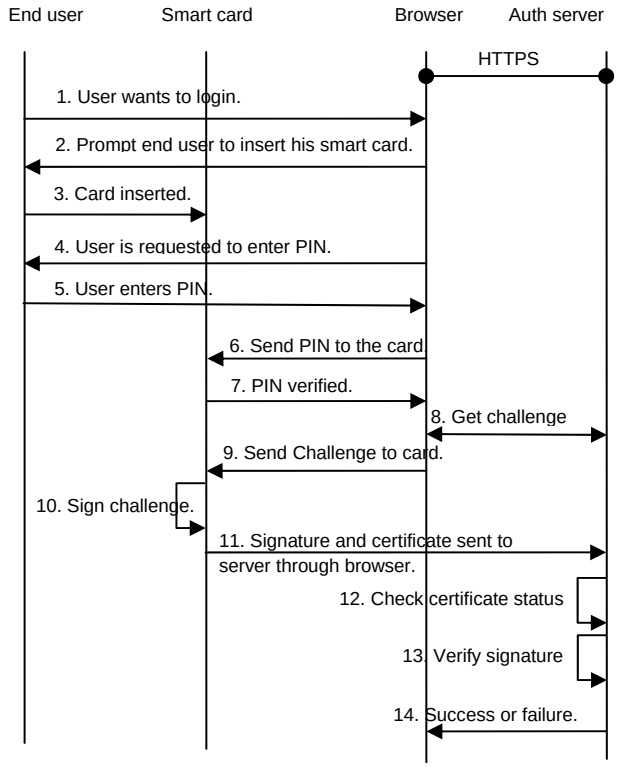

Figure 2. Authentication sequence.

1. The following details the steps in the above sequence:The user clicks the “Login” link in the web page.
2. The authentication client, running in the browser, prompts the user to insert his smart card into a smart card reader attached to the host computer.
3. The user inserts his smart card into the smart card reader.
4. The authentication client prompts the user to enter his PIN in order to use the smart card.
5. The user enters his PIN to the smart card through the web browser user interface or a hardware PIN pad.
6. The authentication client sends the user PIN to the smart card using SConnect communication link.
7. The smart card verifies the PIN, and sends success or failure status back to the browser.
8. If the PIN verification is successful, the client sends a HTTP request to get a challenge from the server. The server responses with a random challenge C that consists of a random number and the server’s domain.
9. The authentication client sends the challenge to SConnect browser plug-in. The latter compares the domain in the challenge with that of the web server to which the browser session is connected. If the two domains are different, SConnect rejects the connection to the smart card. If the two domains are the same, SConnect sends the challenge C to the smart card.
10. The smart card digitally signs the challenge using its private key.
11. The smart card then sends this signature, R, and its X.509 certificate back to the authentication client, which forwards this information to the authentication server.
12. The authentication server verifies the certificate, its issuer, and its revocation status.
13. It also verifies the signature response, R, sent from the card using the public key embedded in the X.509 certificate.
14. If all is good, the server sends a success message to the web browser. Otherwise, the server sends a failure message.

This authentication workflow is simple from the user’s perspective. As evident from Figure 2, the user simply inserts his smart card and enters the PIN. All the details of the X.509 challenge-response handshake for user authentication are hidden from the user. Since there is no classical middleware installation involved, the solution deployment is equally simple for service providers. The challenge for organizations currently using middleware-based smart card authentication solutions is to decide if or not to replace the system with this new approach.

Performance-wise, once the user has inserted the smart card, the time for login is comparable with username/password login, because loading the post-login page typically consumes most of the time.

This proposed method is a two-factor authentication: the what-you-have factor in the form of smart card token (step 3); and the what-you-know factor in the form of user PIN (step 5). It verifies that the user’s smart card indeed holds the private key corresponding to the X.509 certificate. This challenge-response mechanism proves the identity of the user, not whether he has an account at a particular website. The binding of this user identity to a particular account and its access through a given web session are left at the discretion of the service provider web portal.

VI. SECURITY AND USABILITY ANALYSIS

We have presented an authentication framework that is significantly different from the middleware-based authentication architecture used by conventional smart cards. In this section we analyze this framework with respect to protocol security and user behavior.

A. Protocol Security

Our authentication method allows a user to login to a remote web server by proving his digital identity to the server using his smart card. From the security perspective, the authentication relies on four complimentary steps to authenticate the user to the server:

 ISBN: 978-1-61208-152-6

1. Server authentication during SSL handshaking, which is done by the web browser.
2. Server verification done by SConnect.
3. Connection key verification done by SConnect.
4. Certificate-based client authentication (as the user authentication).

The first three steps represent a layered approach to authenticate the server. This ensures that the user is interacting with the intended server. Step 1 validates the server's SSL certificate and establishes a secure communication channel with the server. Step 2 is an additional check on Step 1, in case the user ignores the browser warning about an invalid certificate. Step 3 ensures that smart card connectivity is only exposed to websites with valid connection keys. If a website satisfies these three security checks, SConnect allows it to communicate with the smart card. This significantly reduces the attack surface that a typical web application is subjected to. We get the benefits of an open application development model with easy on demand deployment, but without the risk of MITM and other attacks, which we will discuss in more detail below.

Man-In-The-Middle

As the name suggests, Man-in-the-Middle (MITM) acts as a middle person on the network, intercepting messages between a server and a client to gain access to a user's account at the server. For example, the attacker poses as a server S to an unsuspecting client, and then impersonates as the same client to the actual server S. MITM attacks are typically handled through SSL mutual authentication. The smart card stores the client certificate and the corresponding private key. The web browser has a direct access to the client certificate and the operations that use the private key. The smart card specific middleware discussed earlier in Section III-A makes such access possible. While this approach certainly provides a more robust security model, it is at the expense of usability.

In our proposed authentication method, client authentication is done at the application level, after the SSL handshake and SConnect have verified the server. Although this two-step approach to mutual authentication by itself is vulnerable to MITM, the potential risks are mitigated by the security checks performed by SConnect. Assume an attacker, a malicious website www.bad.com, is acting as MITM between a client and a legitimate server www.good.com through, for example, DNS poisoning or some other means. This MITM attack is addressed as follows:

1. SConnect server verification will fail since the browser has connected to www.good.com, while the common name in SSL server certificate is www.bad.com. SConnect will catch this mismatch even if the user has ignored the browser warning.
2. SConnect uses the web browser's root certificate store to verify the validity of an SSL certificate. In case the attacker uses a self-signed certificate whose issuer certificate is not in the root certificate store, or uses a fake certificate for www.good.com, SConnect server verification will catch the error because it cannot verify the certificate. It will reject the connection even if the user has ignored the browser warning.
3. In the unlikely event that the attacker obtains a valid SSL certificate issued by a trusted CA in the name of www.good.com and the corresponding private key, SConnect will refuse access to smart card unless the attacker also presents a valid Connection Key. The attacker may copy the Connection Key issued to the actual www.good.com, but it still cannot pass the Connection Key verification because the thumbprint of its SSL certificate is different from that of the fake www.good.com SSL certificate.

This layered approach to security allows our authentication framework to offer mutual authentication using a two-stage protocol. While it may not be as secure as a monolithic mutual authentication protocol such as SSL, it offers an excellent balance between security and convenience.

Chosen Protocol Attack

In a chosen protocol attack, the attacker lures the user into using his authentication credential at a malicious website when the same credential can be used on a legitimate website [12]. For example, a user has an account at an online bank, www.bank.com, which supports the smart-card-based authentication. An attacker could lure the user into authenticating to a malicious website using the same smart card. During the authentication process, the attacker can login to the user's account at the bank by forwarding the challenge from the bank to the smart card and the response from the smart card to the bank. In order to do so, the attacker must have a valid SSL certificate and a valid Connection Key. This could happen if an otherwise legitimate website either turns malicious or is temporarily compromised.

The domain information added to the server challenge will prevent such an attack. For example, the domain name www.bank.com is a part of the server challenge for user authentication. The attacker website forwards the challenge to the smart card. There are two possibilities. First, if the attacker changes the domain name to its own, SConnect verification will pass and the smart card will generate a response. However, the response verification on www.bank.com server will fail and so will the authentication. This is because the domain name in the response is different from the actual domain name of www.bank.com. Second, if the attacker does not change the domain name, the SConnect verification will fail because the domain name in the challenge is different

 ISBN: 978-1-61208-152-6

from the current domain. SConnect will reject the connection to the smart card.

B. User Behavior

Modern web browsers check the SSL server certificate when establishing an HTTPS connection with a given website. The purpose of this check is to ensure that the SSL certificate is issued by a trusted certificate authority, the certificate's Common Name (CN) matches the website's URL, and that the certificate has not expired. If any of these assertions fails, the browser informs the user that the certificate is not valid and recommends that user not connect to the given website. However, this is only a recommendation, and browsers will still proceed with the connection if the user ignores this warning. Surprisingly, such warnings are not rare. A survey of Internet use published in 2007 found that roughly two-third of all SSL certificates used for secure connections generated warnings [13]. No wonder users have become accustomed to seeing these SSL warnings and casually ignore them. Our authentication framework uses SConnect server verification to strictly enforce what browsers merely recommend.

VII. CONCLUSIONS

Achieving usable security is very challenging. While smart cards offer unparalleled hardware security, their applicability has been restricted to tightly controlled environments where smart card infrastructure can be managed. This paper introduced a new online authentication framework that is different from other smart-card-based authentication solutions. It works within the existing smart card infrastructure, but still offers a truly plug-n-play experience users have come to expect from web applications. Instead of relying on pre-installed middleware, the new framework uses a browser based approach to access smart card services. This not only provides a more familiar user interface, but also allows online service providers to deploy and update their service offerings without requiring the user to install a new application. Our future work for enhancing this authentication framework will focus on practical issues such as; Connection Key revocation, management of SConnect browser extension updates, and handling of SConnect browser extensions that are issued by different authorities.

With this technology, we foresee a new trend in the development of smart card based Internet security solutions that go well beyond user authentication. Additional security services such as email encryption, document signature, secure transactions, etc. can be delivered on demand using the familiar web browser interface.

REFERENCES

[1] "Gmail, Yahoo Mail join Hotmail; passwords exposed", ComputerWorld, http://www.computerworld.com/s/article/9138956/Microsoft_confirms_phishers_stole_several_thousand_Hotmail_passwords. (last access 07/13/2011).

[2] Byron Acohido, Hackers breach Heartland Payment credit card system. USA Today, http://www.usatoday.com/money/perfi/credit/2009-01-20-heartland-credit-card-security-breach_N.htm, January 2009. (last access 07/13/2011).

[3] Carrie-Ann Skinner, One-Third Use a Single Password for Everything, PCWorld, http://www.pcworld.com/businesscenter/article/161078/onethird_use_a_single_password_for_everything.html, March 2011. (last access 07/13/2011).

[4] Frederik Mennes, Best Practices for Strong Authentication in Internet Banking, ISSA Journal, December 2007. http://www.issa.org/Library/Journals/2007/December/Mennes-Best%20Practices%20for%20Strong%20Authentication%20in%20Internet%20Banking.pdf. (last access 07/13/2011).

[5] Smart Card Alliance, "What makes a smart card secure?," A Smart Card Alliance Contactless and Mobile Payments Council White Paper, CPMC-08002, October 2008. http://www.smartcardalliance.org/resources/lib/Smart_Card_Security_WP_20081013.pdf. (last access 07/13/2011).

[6] Microsoft, Cryptographic Service Providers, http://msdn.microsoft.com/en-us/library/ms953432.aspx. (last access 07/13/2011).

[7] RSA Laboratories, PKCS#11: Cryptographic Token Interface Standard, http://www.rsa.com/rsalabs/node.asp?id=2133. (last access 07/13/2011).

[8] Apple, Mac OS X Security Framework.

[9] C. Adams and S. Lloyd, Understanding PKI: concepts, standards, and deployment considerations, Addison-Wesley Professional; 2nd edition, Nov. 2002.

[10] Kapil Sachdeva, H. Karen Lu, and Ksheerabdhi Krishna, "A browser-based approach to smart card connectivity," IEEE Workshop on Web 2.0 Security and Privacy, Oakland, California, May 21, 2009.

[11] CA/Browser Forum, Guidelines for the Issuance and Management of Extended Validation Certificates, Version 1.0, 7 June 2007, http://www.cabforum.org/EV_Certificate_Guidelines.pdf. (last access 07/13/2011).

[12] R. Anderson, Security Engineering, 2nd edition, Wiley Publishing, Inc., 2008.

[13] C. Jackson and A. Barth, ForceHTTPS: protecting high-security web sites from network attacks, Proceedings of WWW 2008, Apr. 2008, Beijing, China.

 ISBN: 978-1-61208-152-6