Minimizing SSO Effort in Verifying SSL Anti-phishing Indicators

Yongdong Wu, Haixia Yao and Feng Bao

Abstract In an on-line transaction, a user sends her personal sensitive data (e.g., password) to a server for authentication. This process is known as Single Sign-On (SSO). Subject to phishing and pharming attacks, the sensitive data may be disclosed to an adversary when the user is allured to visit a bogus server. There has been much research in anti-phishing methods and most of them are based on enhancing the security of browser indicator. In this paper, we present a completely different approach of defeating phishing and pharming attacks. Our method is based on encrypted cookie. It tags the sensitive data with the server's public key and stores it as a cookie on the user's machine. When the user visits the server so as to perform an on-line transaction, the sensitive data in the cookie will be encrypted with the stored public key of the server. The ciphertext can only be decrypted by the genuine server. Our encrypted cookie scheme (ECS) has the advantage that the user can ignore SSL indicator in the transaction process. The security is guaranteed even if the user accepts a malicious self-signed certificate. This advantage greatly releases user's burden of checking SSL indicator, which could be very difficult even for an experienced user when the phishing attacks have sophisticated vision design.

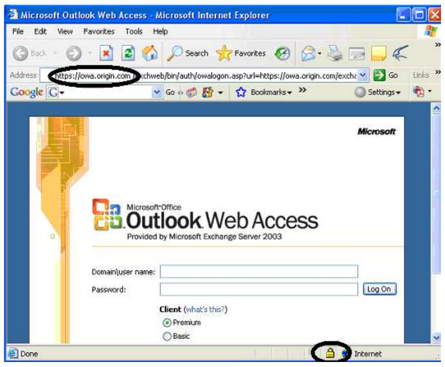
1 Introduction

With the rapid development of Internet and web technologies, most of the online applications such as e-banking and e-government are built on or assisted by WWW. For example, after China Government endorsed the "Digital Signature Law" in 2005, more and more China citizens open e-banking accounts (over 60 million in 2007) such that the transaction amount is in-

Cryptography and Security Department, Institute for Infocomm Research, Singapore
e-mail: \{wydong,hxyao,baofeng\}@i2r.a-star.edu.sg

creased at a rate of 30% annually [1]. Usually, an on-line transaction is built based on client/server model. When a user initiates a transaction with a web browser (*e.g.*, Internet Explorer or Firefox), she will send a request to the web server with its URL (Universal Resource Locator). On a request, the server sends back a form to request the user's personal information. Once the server authenticates the browser with the user's input, the web server sends the confidential page within the browser window which will be shown to the user.

Most web sites currently authenticate users with a simple password submitted via an HTML form similar to Fig.1. Because the personal data are involved, information submitted via HTML forms has proven to be highly susceptible to phishing [2, 3] which exploits the visual mistake to lure victims. Evidence suggests somewhere 3-5% of phishing targets disclose sensitive information and passwords to spoofed web sites [4, 5]; about two million users gave information to spoofed websites resulting in direct losses of $1.2 billion for US banks and card issuers in 2003 [6]. Presently, phishing attacks are becoming more and more sophisticated [7]. As a result, good phishing techniques can fool even the most vigilant users [8, 9], and the countermeasures such as trusted paths [10, 11] and multi-factor authentication schemes [12] are susceptible to phishing.


Microsoft Outlook Web Access - Microsoft Internet Explorer

Address: https://owa.origin.com/exchweb/bin/auth/owalogon.asp?url=https://owa.origin.com/exch

Domain\user name:

Password:

Client (what's this?) [✓ Premium] Basic

Fig. 1 HTTPS GUI of Microsoft Internet Explorer (IE) for self-signed certificate. Firefox will highlight the background of URL status besides those in IE.

To understand the workflow of phishing, let's recall that an on-line transaction has three links which know the personal data: server, transmission channel and user/browser. Hence, to provide secure transaction, the security of each link should be ensured. Nowadays, the server usually installs heterogeneous firewalls, IDS (Intrusion Detection System) *etc* so as to guarantee the safety of the server; meanwhile, the transmission channel is managed with SSL (Secure Socket Layer [13]) protocol which is secure enough for message transmission in an on-line transaction from the viewpoint of a cryptographer. To provide a trustworthy browser/user interface, the browser renders some security indicators so as to provide the user information about the authenticity of the website. Concretely, a browser provides user interface which includes at least three indicators: a HTTPS other than HTTP is shown in the URL bar, besides the target domain name; (2) A closed lock is shown in the status bar if the SSL protocol is performed successfully and the URL bar matches the certificate of server, both (1) and (2) are highlighted with circles in Fig.1; (3) If the user clicks on the closed lock, the visited server certificate will be shown in a pop-up window. If and only if all of the above items are checked carefully, all the links are protected and thus the target server is authentic in principle.

Nonetheless, the transaction scheme may still be vulnerable in practical since the client/browser interface can be easily reproduced with "Web spoofing" technique [14, 15, 16]. Friedman *et al.* [17] summarized that it is difficult for average users to determine whether a browser connection is secure due to the follows:

- It is trivial to insert a spoofed image with any security indicator where one does not exist [14].
- Many users do not understand the meaning of the SSL security indicator. Hence, they ignore the security indicator such that a non-SSL malicious website is mis-regarded as a trusted server.
- Many users do not notice the absence of SSL lock icon.
- The lock icon only indicates that the current page was delivered to the user securely with SSL protocol. However, the page content including the user input can be sent to other website insecurely, or is accessible to other frame shown in the same browser instance in a multi-frame page [18].
- Regardless certificate is critical in verifying the authenticity of the server, few common users understand SSL certificates, and rare users go through the effort of checking SSL certificates and certificate authorities (CAs). Indeed, there are too many cryptology jargons in the definition of digital certificate.
- Some browsers provide warnings to inform the user when data is submitted insecurely, or server certificate is problematic such as expiry or self-signed, but many users ignore these warnings or turn them off.

In summary, the major reason that an attacker can start phishing or pharming attack is that users do not reliably notice the absence of a security indicator, and do not know how to use. Hence, the browser must provide ease of use interface, and minimize the effort in checking the security indicators. In contrast, if an anti-pharming solution is too complicated, it will likely be misused by average users.

This paper presents an anti-phishing scheme called as ECS which enhance the security of the user/browser link. ECS upgrades the browser with handling cookies so as to minimize the user effort in on-line transaction. To this end, the cookie including the password as well as the public key of the target server is generated in the user registration process. To perform an on-line transaction, the browser builds an SSL channel with the server, then the browser merges the password with SSL session key, and encrypts them with the stored public key. The ciphertext is sent to the target server as an encrypted cookie. After the SSL server decrypts the ciphertext, it will recover the password since it knows the session key. This improvement enables that a user is free from checking SSL indicators at any time except the registration period. As a result, the present protection scheme provides to the user the following advantages:

- free from identifying closed lock;
- free from identifying HTTPS and URL in the status bar;
- free from identifying the certification;
- free from identifying self-signing certificate.
- free from dictionary attack.

The organization of the paper is as follows. Section 2 introduces the preliminaries. Sections 3 describes the present scheme. Section 4 addresses our implementation. Section 5 analyzes relevant work. Section 6 concludes the paper.

2 Preliminaries

2.1 Phishing and Pharming

In a phishing attack, an adversary duplicates a known site of business (*e.g.*, `www.paypai.com` mimics `www.paypal.com`) and then sends spams to encourage users to visit the malicious site. When users click on the link within the spam email, they are taken to the fake site to divulge critical information.

Pharming accomplishes the same thing as phishing, but with more stealth and without spam email. In this case, the adversary plants false code on the domain name server (DNS) itself, so that anyone who enters the correct website address will be directed by the DNS to the fake site.

2.2 HTTPS-enabled Browser

Of all security techniques against Internet attacks, SSL3.0 [13] is the *de facto* standard for end-to-end security and widely applied to do secure transactions such as Internet banking. When the client's web browser makes a connection to an SSL-enabled web server over HTTPS, the browser must verify the server's certificate, all the CA's certificates, name of the server certificate against URL status, and expiry. If any of these checks fails, the browser warns the user and asks the user if it is safe to continue. If the user chooses positively, she may permit the SSL connection to continue even though any or all of these checks have failed [19], expiry or self-signed certificates. In reality, researchers have shown that users routinely ignore such security warnings [20, 21, 22]. Unfortunately, this kind of ignorance enables the phishing attack. In other words, SSL merely guarantees that the received message is authentic and confidential in the transmission, but it does not care about the message before or after transmission.

2.3 HTTP Cookie

HTTP cookies (see `http://en.wikipedia.org/wiki/HTTP_cookie`) are parcels of text sent by a server to a web browser and then sent back unchanged by the browser each time it accesses that server. Since cookies may contain sensitive information (user name, a token used for authentication, etc.), their values should not be accessible to other computer applications.

A cookie contains the name/value pair, an expiration date, optional secure flag and version number, a path/domain name. The name/value pair is the content of the cookie; the expiration date tells the browser when to delete the cookie; if the secure flag is set, the cookie is intended only for encrypted connections; the path/domain tells the browser where the cookie has to be sent back. For security reasons, the cookie is accepted only if the server is a member of the domain specified by the domain string. Therefore, a cookie is actually identified by the triple name/domain/path, not only the name. In other words, same name but different domains or paths identify different cookies with possibly different values. Generally, the browser objects including cookie are under control of same origin policy.

2.4 Same Origin Policy

The same-origin policy is an important security measure in modern web browsers for client-side scripting (mostly JavaScript)[1]. It governs access control among different web objects and prevents a document or script loaded from one "origin" from getting or setting properties of a document from a different "origin", where an "origin" is defined using a tuple <domain name, protocol, port>.

[1] Internet Explorer uses an alternative approach of security zones in addition to the same-origin (or "same-site") policy.

3 The Encrypted Cookie Scheme

3.1 Attack Model

Fig.2 illustrates the participants involved in on-line transaction model: user, SSL server, client/browser and attacker. The user will authenticate herself with SSO (Single-Sign-On) to the SSL server, while the SSL server authenticates itself with a certificate issued by a certificate authority. The browser is an application such as Firefox^{TM} or Microsoft Internet Explorer^{TM} which helps the user to make transactions. When a user requests a secure page, the browser will verify the server's certificate with HTTPS/SSL protocol. If the server is authentic, both sides will negotiate a session key for the secure communication, and the user's browser status line shows a security lock. Additionally, if the user clicks the security lock, a popup window will show the security information on server certificate. The attacker aims to impersonate an innocent user by forging an SSL site and luring an innocent user to disclose sensitive data.

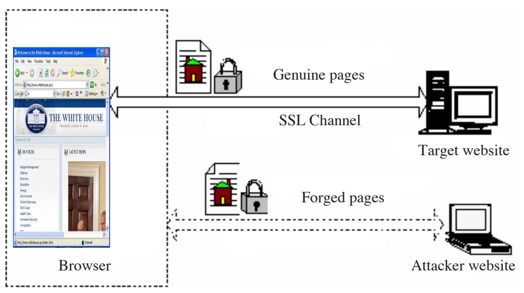


Fig. 2 Attack model

In this paper, suppose that an adversary can create an arbitrary self-sign certificate, but the user ignores the security warning and accepts the certificate in any transaction period. Thus, the attacker can insert, delete and tamper the communication data at will. This phishing attack is powerful such that many countermeasures are invalid. For example, SecurID [23] provides One-Time Password for two-factor authentication and has been deployed in a lot of financial institutions, but it is still vulnerable to the phishing attack.

3.2 ECS Modules

Suppose a server generates a public/private key pair PK/SK, and obtains a certificate C_A from a CA (*e.g.*, `www.verisign.com`) whose public key is hard-coded in the user's browser. To improve the browser's security against phishing attack, ECS encrypts the user's sensitive data with server's public key PK and transmits the encrypted cookie to the server. To this end, ECS has 3 modules: Registering cookie, reading cookie, and verifying cookie.

3.2.1 Registering cookie

After the browser builds an SSL channel with the registration server, it displays a form as Fig.1 for registration. Before filling in the form, a user should carefully verify all the security indicators, *i.e.*, closed lock, HTTPS URL status and attributes of server certificate (subject, issuer, expiry, etc). If all the SSL indicators are in place, the user will input her personal data <`username`, `password`>, and send the complete form to the server. Then the server will return a cookie whose content as

$$C = PK \parallel m \parallel Aux, \tag{1}$$

where $m =$<`username`, `password`>, Aux is the auxiliary information such as certificate version of the server in case of certificate upgrading. Moreover, the server saves `username` and the hash of `password` into a database $\mathbb{B}$.

3.2.2 Reading Cookie

As we mentioned in the Section 2.4, same origin policy makes sure that a cookie is read only when the requested source is the same as the stored one. If a user visits a HTTPS website whose domain matches the cookie domain, the browser will read the cookie after setting up an SSL channel with the transaction server. Then the browser will encrypt the `username/password` pair to generate a cookie whose content is

$$c = \mathcal{E}(PK, m \oplus k \oplus C_A, r), \tag{2}$$

where r is a random number, k is the the SSL session key, and $\mathcal{E}(PK, \cdot)$ is a CCA2 encryption algorithm (*e.g.*, ElGamal cryptosystem). Afterwards, the browser generates cookie $\mathbf{C}$, further sends to the server the cookie as an SSL traffic, *i.e.*, cookie's encryption $\mathcal{F}(k, \mathbf{C})$, where $\mathcal{F}(\cdot)$ is a symmetric cipher such as AES such that no adversary can eavesdrop the traffic.

Remark: a client can not distinguish an adversary from a genuine server by comparing the received public key with the stored public key PK in the transaction since a genuine server may update its public key (ref. Subsection 4.3) from time to time.

3.2.3 Verifying cookie

After receiving the encrypted cookie $\mathcal{F}(k, \mathbf{C})$, the server decrypts it to obtain cookie content c. Then it calculates $m = \mathcal{D}(SK, c) \oplus k \oplus C_A$ with the decryption algorithm $\mathcal{D}(\cdot)$ and its private key SK. Based on the user database $\mathbb{B}$, the server can verify the identity of the user.

3.3 Security Analysis

Based on same origin policy, a cookie will not be read from the user's machine unless the transaction URL domain is identical to the registration URL domain. Hence, to launch a phishing attack, the following diagram should be employed.

- an attacker $\mathcal{A}$ forges a website with the same URL as the genuine site;
- $\mathcal{A}$ selects a public/private key pair, and self-signs his public key to create a certificate $\tilde{C}_A$. Please note that public key of $\mathcal{A}$ must be different from that of a genuine server, otherwise, $\mathcal{A}$ can not setup SSL channel with the user;
- $\mathcal{A}$ lures a user to visit the bogus site. For example, by sending spam email;
- The user visits the bogus site and ignores the certificate warning.

To obtain the sensitive data of a user so as to impersonate her with the above diagram, an adversary $\mathcal{A}$ will start man-in-the-middle attack as follows,

- attacker $\mathcal{A}$ produces a bogus certificate $\tilde{C}_A$, and sends a polynomial number n_1 of queries to the client so as to obtain the ciphertext

  $$c_i = \mathcal{E}(PK, m \oplus k_i \oplus \tilde{C}_A, r_i), i = 1, 2, \ldots, n_1,$$

  where k_i is the SSL session key, and r_i is random.
- attacker $\mathcal{A}$ tries to impersonate the user by connecting to the genuine server. Both $\mathcal{A}$ and server negotiate an SSL session key k with the server. Clearly, $k \oplus C_A \neq k_i \oplus \tilde{C}_A$ for any $i \in [1, n_1]$
- attacker $\mathcal{A}$ continues to send a polynomial number n_2 of queries to the user so as to obtain the ciphertext

  $$c_j = \mathcal{E}(PK, m \oplus k_j \oplus \tilde{C}_A, r_j), j = n_1 + 1, \ldots, n_1 + n_2,$$

  where k_j is the SSL session key, r_j is random, and $k \oplus C_A \neq k_j \oplus \tilde{C}_A$ for any $j \in [n_1 + 1, n_1 + n_2]$.

Since $\mathcal{E}(\cdot,\cdot)$ is CCA2, it is semantically secure such that the distribution of $\mathcal{E}(\cdot,\cdot)$ is uniform for any m. Thus,

$$\mathcal{H}(m) = \mathcal{H}(m \mid c_i, k_i \oplus C_A), i = 1, 2, \ldots, n_1 + n_2, \tag{3}$$

where $\mathcal{H}(X)$ represents the entropy of variable X. Informally, m is independent from c_i due to the random number r_i, thus the query results provide negligible information to adversary $\mathcal{A}$ in generating an encryption $c = \mathcal{E}(PK, m \oplus k \oplus C_A)$. Therefore, to impersonate a user, $\mathcal{A}$ has to generate a well-formed ciphertext $c = \mathcal{E}(PK, m \oplus k \oplus C_A, r)$ from (PK, k, C_A) and some r but without knowing m.

If $\mathcal{A}$ succeeds in generating c at non-negligible probability, $\mathcal{A}$ queries the server with c so as to obtain $m \oplus k \oplus C_A$. Thus $\mathcal{E}(PK,\cdot)$ is not secure against chosen ciphertext attack, *i.e.*, $\mathcal{E}(PK,\cdot)$ is not CCA2. It contradicts with our assumption on $\mathcal{E}(PK,\cdot)$.

On the other hand, Eq.(2) demonstrates that the present scheme is secure against dictionary attack.

4 ECS Implementation

In our implementation, Firefox browser (`http://developer.mozilla.org/en/docs/Download_Mozilla_Source_Code`) and Apache server are used as testbed for demonstrating ECS. A module in Firefox is used to encrypt `<username, password>` to generate an encrypted cookie in the reading stage, and a PHP program is used to generate cookie and verify the encrypted cookie.

4.1 Registration Process

The registration is performed for the first time when a user visits an SSL server without a cookie. In order to make use of on-line transaction, users usually obtain sensitive data such as password via out-of-band channel (*e.g.*, face-to-face delivery, mail) in advance. To register a user on line, the server will send a form Fig.1.

After a user checks all the security indicators (*i.e.*, lock, URL,certificate), she fills in the form and submits the complete form to the SSL-server. Afterwards, the client will receive an HTTPS page generated with code segment in Fig.3 from the SSL-server. When the browser cookie is enabled, the new cookie will be stored in the user site.

```
<?php
$value = C;
$expiry=3600× 24× 365;
setcookie("ECS_USER", $value, time()+$expiry, "/", "192.168.137.211", 1);
?>
```

Fig. 3 Setting up an example cookie. The cookie name is ECS_USER, its value is the string C which is generated in Eq.(1), its expiry is of 365 days. The path/domain `/192.168.137.211` tells the browser to send the cookie when requesting a page of the domain `192.168.137.211`.

4.2 Transaction Process

After registration, a cookie which including the `username/password` and server public key is stored in user's machine. If a user likes to make a transaction, it is unnecessary to input her password any more, *i.e.*, Fig.2 will not be shown in the transaction period. With regard to Fig.4, the cookie will be read when the user visits the authentic server based on the same origin policy, and processed according to the proposed scheme in Section 3. Afterwards, the encrypted cookie is sent to the server for verification. Concretely,

- In the member function *nsCookieService::GetCookieInternal*() of the source file *mozilla\netwerk\cookie\src\nsCookieService.cpp*, we add ECS code for searching the cookie with name "ECS_USER", reading the cookie value, parsing the value according to Eq.(1), and generating c with Eq.(2).
  Moreover, since either IETF RFC2109 or cookie processing module uses semi-colon (`0x3B`), addition (`0x2B`), comma (`0x2C`), LF (`0x0A`) and NUL (`0x00`) as separators, and the ciphertext c may have the separators, ECS encodes c such that these separators do not occur in cookie. Technically, As shown in the following table, ECS scans ciphertext c byte by byte, and replaces byte "\" (or ";" ,"+", ",", LF, NUL) with two bytes "\\" (or "\A", "\B","\C","\D", "\E" respectively).

Original symbol(ASCII code)		Coded symbols
backslash (`0x5C`)	$\leftrightarrow$	"\\" (`0x5C5C`)
semi-colon (`0x3B`)	$\leftrightarrow$	"\A" (`0x5C41`)
addition (`0x2B`)	$\leftrightarrow$	"\B" (`0x5C42`)
comma (`0x2C`)	$\leftrightarrow$	"\C" (`0x5C43`)
LF (`0x0A`)	$\leftrightarrow$	"\D" (`0x5C44`)
NUL (`0x00`)	$\leftrightarrow$	"\E" (`0x5C45`)

- At the server side, after receiving the cookie, the server decodes the value to ciphertext c by replacing two bytes "\\" (or "\A", "\B","\C","\D", "\E") with one byte "\" (or ";" , "+", ",", LF, NUL resp.) sequentially, then decrypts c with its private key. If the decrypted password matches the stored one, the user is authentic.

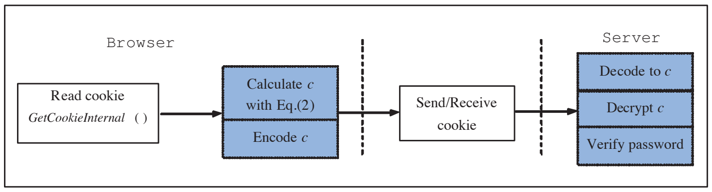


Fig. 4 User authentication process. The shadow units are developed for ECS.

4.3 Refreshment Process

When the server receives the cookies from the client, it will check the version, and select the private key based on the version. The private key enables the server to decrypt the ciphertext correctly. If the `username/password` matches a registration record of database $\mathbb{B}$, the user is authenticated to perform on-line transaction. Furthermore, if the version is not latest, the server will send a new cookie $\tilde{c}$ with Eq.(1) including the old password, new public key and new version. If the new cookie $\tilde{c}$ has the same password as that of the old cookie c, the client/browser will replace the old cookie with the new one.

5 Related work

Since phishing attack is a realistic risk in on-line transaction, there are many countermeasures on phishing attacks based on different security models.

5.1 Manual-checking Mechanism

Synchronized Random Dynamic (SRD) scheme [27, 28] defines an internal reference window whose color is randomly changed, and sets up the boundary of the browser window with different colors according to certain rules. If the boundary of a pop-up window de-synchronizes with that of the reference window, the user concludes that a web-spoof attack is under way. However, it is impractical for the device of small screen (such as hand-held device) because it is inconvenient to open two windows and switch between the windows. Moreover, the attacker can create a bogus reference window to overlap the original reference window.

RFC2617 [29] proposes a Digest Access Authentication scheme which employs password digest to authenticate a user. PwdHash [11] authenticates a user with a one way hash of the tuple <password, domain name> instead of password only so as to defeat the visual mistake on URL. Moreover, Dynamic

Security Skins (DSS) [30] creates a dedicated window for inputting username and password so as to defeat bogus window. After both server and client will negotiated a session key, a remote server generates a unique abstract image called as “skin” that automatically customizes the browser window or the user interface elements in the content of a remote web page. Similarly, the client browser independently computes the same image. To authenticate content from an authenticated server, the user needs to perform one visual matching operation to compare two images. In addition, since `username` should be disclosed to the server before authentication, `username` will be known to the phishing attacker.

Adelsbach *et al.* [31] combines all concepts in an adaptive web browser toolbar, which summarizes all relevant information and allows the user to get this crucial information at a glance. As this toolbar is a local component of the user’s system, a remote attacker cannot access it by means of active web languages. The advantage of this implementation is that a user has a permanent and reliable overview about the status of his web connection. Once a user has personalized the browser’s GUI, users achieve sufficient security against visual attacks. Users only have to verify the web browser’s personalization and the certificate information, which is always displayed. A disadvantage of the toolbar described above is that the user must recognize his personal image at each login.

5.2 Auto-checking mechanism

As an improvement on [31], ADSI (Automatic Detecting Security Indicator) [32] generates a random picture and embeds it into the current web browser. It can be triggered by any security relevant event occurred on the browser, and then performs automatic checking on current active security status. When a mismatch of embedded images is detected, an alarm goes off to alert the users. Since an adversary is hard to replace or mimic the randomly generated picture, the web-spoofing attack can not be mounted. However, ADSI can not prevent man-in-the-middle phishing attack with self-sign certificate.

Adida [33] presents a FragToken scheme which employs the URL fragment as an authenticator, and change-response for authentication. FragToken is only useful in low-security environment (*e.g.*, Blog) since it is vulnerable to man-in-the-middle attack.

By examining the domain name, images and links, SpoofGuard [34] examines web pages and warns users when a certain page has a high probability based on the blacklist in the server site.

Cache Cookie [35] utilizes the browser cache files to identify the browser. It does not install any software into the client side and hence is easy of deployment. Another cookie based scheme is called Active cookie scheme [36] which stores both the user’s identification and a fixed server IP address.

When a client visits the server, the server will redirect the client to the fixed IP address. In short, Active cookie scheme acts as replacing URL domain name with IP address so as to defeat pharming attack.

Karlof *et al.* [18] proposed the locked same-origin policy (LSOP) enforces access control for SSL web objects based on servers' public keys. LSOP grants access only if the stored public key is identical to the public key sent with a new connection. Applying the locked same-origin policy to SSL-only cookies yields locked cookies, an extension to SSL-only cookies which binds them to the public key of the originating server. However, as pointed out in [18], LSOP does not consider the input problem such as SSO. For example, LSOP is vulnerable to the most popular phishing attack which asks an innocent user to fill in a password/account page as Fig.1.

For comparison, Table 1 lists the security performance of related countermeasure and ECS. It demonstrates that the user effort for transaction security is minimal. The weakness is that ECS asks the client to install a patch in the client browser once. Nonetheless, this one-time installation is worthy for the minimal effort in the transaction in comparison with the tedious certificate management work in client-side SSL scheme.

Table 1 Comparison in terms of client effort in on-line transaction.

	Free from checking URL	Free from checking SSL lock	Free from checking Cert warn	Free from checking GUI	Free from MiMA	Free from installing patch
Pwdhash[11]	✓	✓	×	✓	×	×
Client SSL [13]	✓	✓	×	✓	✓	✓+
LSOP[18]	×	✓	×	×	×	×
SRD[28]	✓	✓	×	×	×	×
DigestAccess[29]	×	×	×	✓	×	✓
DSS[30]	×	✓	×	×	×	×
FragToekn[33]	×	×	×	✓	×	✓
SpoofGuard[34]	×	×	×	✓	×	×
ActiveCookie[36]	✓	×	×	×	×	✓
Present ECS	✓	✓	✓	✓	✓	×

URL: https://domain;
SSL lock: the closed SSL lock on the status bar;
Cert warn: a pop-up window for self-signed/non-signed certificate;
GUI: the browser window against a reference window;
MiMA: man-in-the-middle attack; It has minor difference from general MiMA.
Patch: (source/exectutable) code inserted in browser;
$^+$: Client-side SSL does not install any software patch, but it has to manage certificates with much effort.

6 Conclusions

Users' psychological acceptance of an authentication mechanism is vital to its success [37]. However, users' interpretations of "secure" web connections vary significantly, and many users have trouble accurately interpreting browser security indicators and cues, such as URL bar, locked icon, certificate dialogs, and security warnings [21].

The encrypted cookie scheme ECS minimizes the user effort and guarantees that only the target server can obtain the cookie. It does not modify the access role and protocol but the cookie reading module, hence it is easy for deployment.

References

1. Zhensheng Guan, Invited talk in International ICT Security Exhibition & Conference, Guangzhou, China, Nov. 28, 2007.
2. Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach, "Web spoofing: An Internet Con Game," 20th National Information Systems Security Conference,1997. http://www.cs.princeton.edu/sip/pub/spoofing.html
3. Serge Lefranc, and David Naccache, "Cut and Paste Attacks with Java," http://eprint.iacr.org/2002/010.ps
4. Evgeniy Gabrilovich, and Alex Gontmakher, "The homograph attack," *Communications of ACM*, 45(2):128, 2002.
5. Martin Johns, "Using Java in anti DNS-pinning attacks," http://shampoo.antville.org/stories/1566124/,February2007.
6. Avivah Litan, "Phishing Attack Victims Likely Targets for Identity Theft," in Gartner First Take FT-22-8873. 2004, Gartner Research
7. Microsoft. Microsoft security bulletin MS01-017: Erroneous VeriSign-issued digital certificates pose spoofing hazard,March 2001. http://www.microsoft.com/technet/security/Bulletin/MS01-017.mspx
8. Tyler Close, "Waterken YURL," http://www.waterken.com/dev/YURL/httpsy/
9. Russell Housley, Warwick Ford, Tim Polk, and David Solo, "Internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile," 2002. http://tools.ietf.org/html/rfc3280
10. V. Benjamin Livshits, and Monica S. Lam, "Finding security vulnerabilities in Java applications using static analysis," USENIX Security Sym., pp.271-286, 2005.
11. Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell, "A Browser Plug-in Solution to the Unique Password Problem," Usenix Security Symposium, 2005.
12. mozilla.dev.security, "VeriSign Class 3 Secure Server CA?," http://groups.google.com/group/mozilla.dev.security/browse_thread/threa%d/6830a8566de24547/0be9dea1c274d0c5, March 2007.
13. A. Freier, P. Kariton, and P. Kocher, "The SSL Protocol: Version 3.0," Netscape communications, Inc., 1996.
14. Tieyan Li, and Yongdong Wu, "Trust on Web Browser: Attack vs. Defense," First MiAn International Conference on Applied Cryptography and Network Security, LNCS 2846, pp.241-253, 2003.
15. Jeffrey Horton, and Jennifer Seberry, "Covert Distributed Computing Using Java Through Web Spoofing," ACISP, pp.48-57, 1998. http://www.uow.edu.au/ jennie/WEB/JavaDistComp.ps.

16. F. De Paoli, A.L. DosSantos, and R.A. Kemmerer, "Vulnerability of Secure Web Browsers," National Information Systems Security Conference, 1997.
17. Batya Friedman, David Hurley, Daniel Howe, Edward Felten, and Helen Nissenbaum, "Users' Conceptions of Web Security: A Comparative Study," Conference on Human Factors in Computing Systems, pp.746-747, 2002.
18. Chris Karlof, Umesh Shankar, J.D. Tygar, and David Wagner, "Dynamic pharming attacks and the locked same-origin policies for web browsers," CCS 2007.
19. Security Space and E-Soft, "Secure Server Survey," `http://www.securityspace.com/s_survey/sdata/200704/certca.html`, May 2007.
20. Stephen Bell, "Invalid banking cert spooks only one user in 300," ComputerWorld New Zealand, `http://www.computerworld.co.nz/news.nsf/NL/-FCC8B`
21. Rachna Dhamija, J. D. Tygar, and Marti Hearst, "Why phishing works," SIGCHI Conference on Human Factors in Computing Systems, pp.581-590, 2006.
22. Min Wu, Robert C. Miller, and Simson Garfinkel, "Do security toolbars actually prevent phishing attacks?" the SIGCHI Conference on Human Factors in Computing Systems, pp.601-610, 2006.
23. RSA Security Inc, "SecurID product description," `http://rsasecurity.com/node.asp?id=1156`.
24. Sudhir Aggarwal, Jasbinder Bali, Zhenhai Duan, Leo Kermes, Wayne Liu, Shahank Sahai, and Zhenghui Zhu,"The Design and Development of an Undercover Multipurpose Anti-Spoofing Kit (UnMask)," 23rd Annual Computer Security Applications Conference, 2007.
25. M. Burnside, Blaise Gassend, Thomas Kotwal, Matt Burnside, Marten van Dijk, Srinivas Devadas, and Ronald Rivest, "The untrusted computer problem and camera-based authentication," International Conference on Pervasive Computing, LNCS 2414, pp.114-124, 2002.
26. Pim Tuyls, Tom Kevenaar, Geert-Jan Schrijen, Toine Staring, and Marten van Dijk, "Visual Crypto Displays enabling Secure Communications," Proceeding of First International Conference on Security in Pervasive Computing, pp.12-14, 2003.
27. Yougu Yuan, Eileen Zishuang Ye, and Sean Smith, "Web Spoofing," 2001. `http://www.cs.dartmouth.edu/reports/abstracts/TR2001-409/`
28. Eileen Zishuang Ye, and Sean Smith, "Trusted Paths for Browsers," ACM Transactions on Information and System Security,8(2):153-186, 2005.
29. J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication, June 1999. `http://www.ietf.org/rfc/rfc2617.txt.`
30. Rachna Dhamija, and J.D. Tygar, "The Battle Against Phishing: Dynamic Security Skins," Symposium On Usable Privacy and Security (SOUPS) 2005.
31. Andre Adelsbach, Sebastian Gajek, and Jorg Schwenk, "Visual Spoofing of SSL Protected Web Sites and Effective Countermeasures," Information Security Practice and Experience(ISPEC), LNCS 3469, pp.204-216, 2005.
32. Fang Qi, Tieyan Li, Feng Bao, and Yongdong Wu, "Preventing Web-Spoofing with Automatic Detecting Security Indicator," ISPEC, LNCS 3903, pp. 112-122, 2006.
33. Ben Adida,"BeamAuth: Two-Factor Web Authentication with a Bookmark," CCS 2007.
34. Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, and John C. Mitchell, "Client Side Defense Against Web-based Identity Theft," `http://crypto.stanford.edu/SpoofGuard/#publications`
35. Ari Juels, Markus Jakobsson, and Tom N. Jagatic, "Cache Cookies for Browser Authentication," IEEE Symp. on Security and Privacy, pp.301-305, 2006.
36. Ari Juels, Markus Jakobsson, and Sid Stamm,"Active Cookies for Browser Authentication," `http://www.ravenwhite.com/files/activecookies--28_Apr_06.pdf`.
37. J. H. Saltzer, and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, 63(9):1278-308, Sept. 1975.