# Enhancing the Security of On-line Transactions with CAPTCHA Keyboard

Yongdong Wu, Zhigang Zhao

Institute for Infocomm Research, Singapore
{wydong, zzhao}@i2r.a-star.edu.sg

**Abstract.** In an on-line transaction, a client usually have to present some authenticators (password, user certificate or both) to the server. However, those authenticators are exposed to client-side malware such that the malware is able to obtain the server-client messages, or impersonate the user to build another "secure" channel with the server.
The present paper aims to patch this client-side security flaw with a novel password-input method. Specifically, it enables a user to input a password by clicking an on-screen CAPTCHA keyboard, rather than a keyboard typing. The CAPTCHA keyboard is designed to greatly increase the difficulty of password eavesdropping and phishing in a malicious environment given that the malware can not monitor the browser secret memory space. Our implementation shows that Firwfox browser incorporated with CAPTCHA Keyboard and smartcard is viable and transparent over HTTPS protocol.

## 1 Introduction

Presently, most of the on-line applications such as e-banking and e-government are built on or assisted by HTTPS protocol. These applications carry on the server-client procedure as follows. When a user initiates a transaction with a web browser (*e.g.*, Internet Explorer or IE for short, Firefox), she will send a request to the web server with its URL (Universal Resource Locator). On a request, the server will ask the user to input her personal information. Once the server authenticates the browser with the user's input, the web server sends a confidential page within the browser window which will be shown to the user. In order to carry on the on-line transaction securely, the international standard SSL/TLS [1] allows both server and client to authenticate each other and negotiate a cryptographic suite before transmitting and receiving the first byte of data. In the protocol, mutual authentication between client and server is the most critical component.

All the smartcard-based SSL/TLS schemes assume that the channel between the browser and smartcard is secure. However, this assumption does not always hold when the computer is compromised. *An attacking Trojan horse program that places itself between the signing software and the signature smart card can observe the communication between the two parties*[2]. In a compromised computer, the malware is able to intercept the password input and hence it can access the

smartcard at will such that the above smartcard-based authentication schemes fail.

This paper presents a CAPTCHA keyboard to deter password eavesdropping attack and a watermark mechanism to discourage password phishing. Specifically, in the process certificate-based TLS channel set-up, a broswer will generate an on-screen CAPTCHA keyboard windows for inputting password. The CAPTCHA window must have the same visible watermark (or background) as that of the browser main window. When a user clicks the CAPTCHA keyboard, the click points are mapped to a password by the browser internally. Furthermore, all the communication messages between the smarcard and the browser are protected. The implementation on Firefox shows that the scheme is viable.

## 2 Secure On-line Transaction Protocol

### 2.1 Security model

There are 5 parties in an on-line transaction workflow: user $\mathcal{U}$, remote server $\mathcal{S}$, Internet browser $\mathcal{B}$, user smartcard $\mathcal{C}$, and adversary $\mathcal{A}$. User $\mathcal{U}$ communicates with server $\mathcal{S}$ via the browser $\mathcal{B}$. In order to build a secure channel with SSL/TLS protocol, the server has a certificate issued by a CA (Certificate Authority). Browser $\mathcal{B}$ holds the public key of the CA so as to verify the server's certificate. The user holds smartcard $\mathcal{C}$ which stores a certificate issued by server $\mathcal{S}$ or CA and the corresponding private key. In order to prove the ownership of the smartcard, user $\mathcal{U}$ shares a password $p$ with smartcard $\mathcal{C}$. Besides the user-smartcard password $p$, user may share a password $P$ with server. These two passwords may or may not be the same. The adversary $\mathcal{A}$ has the capability to control all the communication channels. More precisely,

- Adversary $\mathcal{A}$ controls the network between browser $\mathcal{B}$ and server $\mathcal{S}$. He is able to disrupt the communication or alter data at will. For example, an adversary $\mathcal{A}$ can have this capability if he is in a privileged "gateway" of the network.
- Adversary $\mathcal{A}$ is able to install malware in the user's machine so as to take partial control of the user's machine and perform some actions such as monitoring USB communication, Network communication, keystroke monitoring, screen dumping etc.
- Adversary $\mathcal{A}$ is not able to monitor/modify the browser secret space where secrets are stored and manipulated due to software protection technologies such as code obfuscation. Hence, it is beyond our scope to defeat the attack where $\mathcal{A}$ exploits the secrets (e.g., session key, random source), and CA's public key stored inside browser $\mathcal{B}$, etc.
- Adversary $\mathcal{A}$ is unable to corrupt either server $\mathcal{S}$ or smartcard $\mathcal{C}$.
- CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart, [3]) problem can not be solved by computer technology, e.g., OCR recognition technologies.

### 2.2 Security-enhanced TLS protocol

Given that a user shares a password $p$ with a smartcard which stores a certificate $C_u$/private key $pru$, the present browser-smartcard protocol proves that the user owns the smartcard. Furthermore, assume the server has an authorized certificate $C_s$, it builds a secure TLS channel with brower/smartcard. Thus, the user and the server are able to set-up a secure channel even in the presence of malware. As shown in Fig.1, the security-enhanced TLS protocol is as follows.

(1) When a browser is activated, it selects a one-time secret watermark **W**. Whenever a TLS channel is setting up, the watermark **W** is used as the background of the page.
(2) To ensure the authenticity of the smartcard ownership, Browser $\mathcal{B}$ and smartcard carries on a verification protocol as follows:
   - Browser asks the user to input a password $p$ with CAPTCHA keyboard (Subsection 2.3). Then it sends to Smartcard $\mathcal{S}$ a request message `Login` so as to negotiate a session secret $k$ with smartcard. For instance, we can employ Encrypted Key Exchange (EKE) [4] to share an authentic session key $k$ between browser and smartcard such that any original plaintext $m$ between them is encrypted with an authentic symmetric cipher $\mathcal{E}_k(\cdot)$.
   - Browser sends to smartcard a `CertificateVerify` request including the authenticated encryption $\mathcal{E}_k(h)$, where $h$ is the hash value of all the transcripts in the previous steps between browser and server.
   - Smartcard $\mathcal{C}$ recovers $h$ by decrypting the received ciphertext with the session key $k$, and then creates the signature $\sigma = SIGN_{pru}(h)$ with user's private key $pru$ stored in the smartcard. Afterwards, smartcard sends the signature $\sigma$ to Browser.
   - Browser sends to Server $\mathcal{S}$ a `CertificateVerify` message including $\sigma$. Then Server $\mathcal{S}$ verifies the signature using the user's public key $pku$ extracted from user's certificate.
(3) Continue the rest of TLS protocol.

### 2.3 On-screen CAPTCHA keyboard

In the on-line transaction, the smartcard will perform all the operations related to the private key, e.g., decrypting and signing. In order to ensure that only the authorized party can perform these security-related operations, a password should be used, otherwise, a malware can obtain the decrypted data, or genuine signatures for bogus data. Hence, a password is usually required for smartcard applications. However, when a user types the password with a normal keyboard, the password may be easily intercepted by a malware. Although an on-screen keyboard is able to deter the malware, it can be invalidated by an advanced OCR-enabled malware.

We design a new user interface as Figure 2 (left) to input password to browser by adapting the CAPTCHA keyboard [5]. Specifically, when a browser asks for

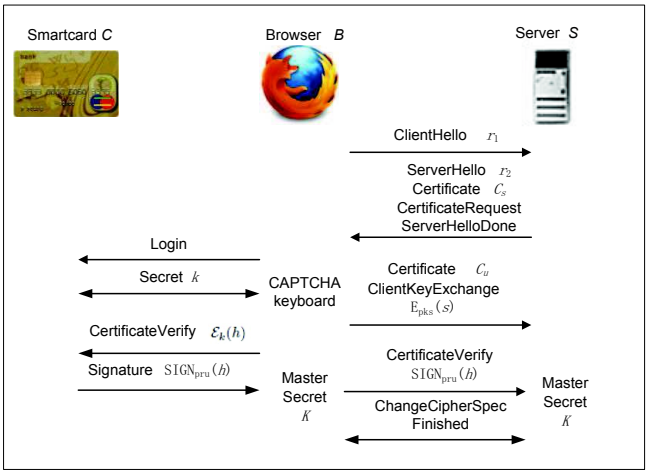


**Fig. 1.** Malware-resistant TLS handshaking protocol in on-line transactions.

inputting a password, it outputs a CAPTCHA keyboard image whose background is the watermark **W**. Before clicking the CAPTCHA keyboard, the user shall check whether the background (or watermark **W**) of the CAPTCHA window matches that of the broswer main window. If not, the user will terminate the transactions. Otherwise, the user clicks the CAPTCHA keyboard, the character in the position is regarded as the input of the password. As the browser creates the image and knows the mapping between the character and position, it can obtain the password. Hence, the on-screen CAPTCHA keyboard has the merits of both text-based CAPTCHA and object-based CAPTCHA: good for password input and secure against keyboard/mouse eavesdropping.

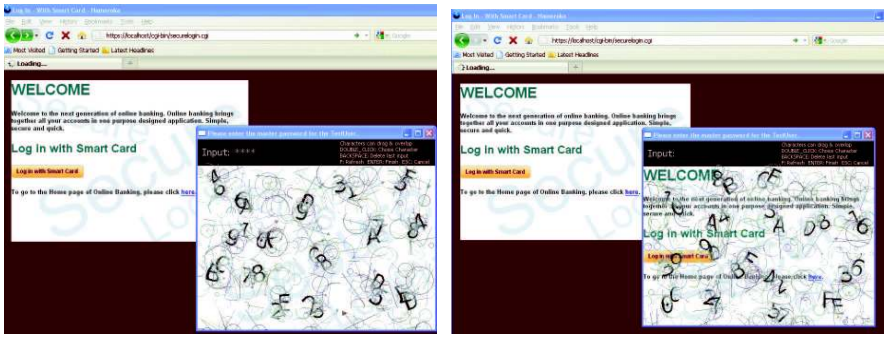


**Fig. 2.** CAPTCHA keyboard. Left: genuine, Right: Bogus.

## 3 Discussion

### 3.1 Security

**Protocol security:** In the process of SSL/TLS handshake, the smartcard is used as a trusted computing platform for client-side authentication. Hence, the setup process of client-server communication channel is secure if the browser environment is trusted. Meanwhile, browser and smartcard run the standard password authentication protocol for login process, hence, the login process is secure too. Hence, an attacker may turn to exploit the browser environment so as to get the secrets of the user, i.e., the private key and/or the password. As the private key is stored in the smartcard, and never disclosed outside the smartcard, the adversary has no means to obtain the private key. But if the adversary is able to get the password, it has the capability to access the private oracle such as signing at will. Thus, an adversary will attempt to eavesdrop the password so as to penetrate the on-line transaction scheme.

**Password phishing:** An adversary may forge a CAPTCHA keyboard and allude the user to disclose her password. However, this phishing attack can be detected by the user because this bogus window will have noticeable artifacts. According to our protocol, a malware does not have the original watermark **W**, it has to dump the screen and super-impose a CAPTCHA keyboard to generate a fake password input window. Therefore, the content in the browser window will exist in the phishing window as Figure 2 (right).

**Password eavesdropping:** As described in Subsection 2.3, a password is input when a user clicks on the CAPTCHA image. In this process, a malware is able to obtain the clicking points by hooking the mouse input routine. However, the malware can not obtain the password because it can not solve CAPTCHA from the points and the CAPTCHA image. The argument is as follows. If the adversary can solve CAPTCHA with the intercepted coordinates, the adversary is capable of recognizing the CAPTCHA at a very high probability when the distorted characters are close to each other. However, this capability is in conflict with CAPTCHA assumption. In addition, the present 2-dimentional CAPCHTA does not require that the characters align in a (rough) line. Indeed, as each character can be randomly distributed over a 2-D image background, and overlapped with each other, the present CAPTCHA mechanism is at least as secure as the conventional text-based CAPTCHA. Hence, the password input with CAPTCHA keyboard is secure even if the malware is able to get the CAPTCHA keyboard and click points.

**Human-assisted eavesdropping:** The present scheme is not a panacea because an adversary may help a malware to get the password. Specifically, a malware may eavesdrop the CAPTCHA screen and click points, then send them to a remote human. Based on the assumption of CAPTCHA, the human is able

to solve the CAPTCHA problem. However, this "successful" attack incurs extra human effort, and high risk due to the trace (e.g., IP address) leakage of the human such that the attacker may not launch this attack.

### 3.2 Performance comparison

With the similar configuration as other browsers, the present scheme provides overall better performance. Table 1 shows the comparison result with the competitors. The second column indicates the requirements for user certificate. The present scheme is applicable to absence/presence of client certificate. In the third column, scheme [7] has additional requirement for the web pages due to the proxy mechanism. The fourth and fifth columns show that only the present scheme is secure in the malicious host, i.e., only the present scheme can provide a secure on-line transaction in a compromised and legacy environment. All of the schemes have the same processing time at both the smartcard and server.

**Table 1.** Performance comparison.

| | User's Certificate | Standard compatibility | Keystroke Anti-eavesdropping | Password Anti-Phishing | Smartcard security |
|---|---|---|---|---|---|
| Microsoft IE | Optional | Yes | No | No | Low |
| TLS+SESAME[6] | Yes | Yes | No | No | Low |
| Urien [7] | Yes | No | No | No | Low |
| OTP | No | Yes | No | No | Nil |
| Present | Optional | Yes | Yes | Yes | High |

## References

1. T. Dierks, and E. Rescorla, "The TLS Protocol, Version 1.1," IETF Draft, RFC 2246, 2005.
2. Adrian Spalka, Armin B. Cremers, and Hanno Langweg, "The fairy tale of What You See Is What You Sign' - Trojan Horse Attacks on Software for Digital Signatures," IFIP Working Conf. on Security and Control of IT in Society-II, 2001.
3. G. Mori, J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," CVPR, vol.1, pp.134-141 , 2003.
4. Y. Sheffer, G. Zorn, H. Tschofenig, S. Fluhrer, "An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol," IETF RFC 6124, 2011.
5. M. Szydlowski, C. Kruegel, and E. Kirda, "Secure input for Web applications," ACSAC, pp.375-384, 2007.
6. Mark Vandenwauver, Paul Ashley, Joris Claessens, Mark Looi, and Wim Moreau, "Using Smart Cards to Integrate SSL/TLS and SESAME," IFIP TC6/TC11 International Conference on Communications and Multimedia Security, vol. 152, pp. 303-317, 1999.
7. Pascal Urien, "Collaboration of SSL smart cards within the WEB2 landscape," Int'l Symposium on Collaborative Technologies and Systems, pp. 187-194, 2009.