SSL/TLS Session-Aware User Authentication: A Lightweight Alternative to Client-Side Certificates

Rolf Oppliger[1] Ralf Hauser[2] David Basin[3]

[1]eSECURITY Technologies
Beethovenstrasse 10, CH-3073 Gümligen
Phone/Fax: +41 79 654 8437
E-mail: rolf.oppliger@esecurity.ch

[2]PrivaSphere AG
Jupiterstrasse 49, CH-8032 Zürich
Phone: +41 43 299 5588, Fax: +41 (0)1 382 2133
E-mail: hauser@privasphere.com

[3] ETH Zurich, Department of Computer Science
Haldeneggsteig 4, CH-8092 Zürich
Phone: +41 44 632 7245, Fax: +41 44 632 1172
E-Mail: basin@inf.ethz.ch

Abstract

Many SSL/TLS-based e-commerce applications employ traditional authentication mechanisms on the client side. These mechanisms—if decoupled from SSL/TLS session establishment—are vulnerable to man-in-the-middle attacks. In this article, we examine the feasibility of such attacks, survey countermeasures, and explain the rationale behind SSL/TLS session-aware user authentication as a lightweight and privacy-enhancing alternative to the deployment and use of public key certificates on the client side. We also present different possibilities for making deployed user authentication mechanismsSSL/TLS session-aware.

Keywords. Security, electronic commerce, man-in-the-middle (MITM) attack, SSL/TLS protocol, user authentication, SSL/TLS session-aware user authentication

1 Introduction

Most electronic commerce (e-commerce) applications in use today employ the *Secure Sockets Layer* (SSL) or *Transport Layer Security* (TLS) [DR06] protocol to authenticate the server and to cryptographically protect the communication channel between the client and the server. SSL/TLS provides support for user authentication based on public key certificates. But in practice, due to the slow deployment of these certificates, user authentication usually occurs at the application layer. There are many options here, including personal identification numbers (PINs), passwords, passphrases, as well as “strong” authentication mechanisms, such as one-time password (OTP) and challenge-response (C/R) systems.

In theory, the SSL/TLS protocol is assumed to be sound and secure. In practice, however, the vast majority of SSL/TLS-based e-commerce applications, employing user authentication at the application layer, are vulnerable to phishing, Web spoofing, and—most importantly—*man-in-the-middle* (MITM) attacks. If an MITM can place himself between the user and the server, then he can act as a relay and authenticate himself to the server on the user's behalf. Even worse, if the MITM operates in real-time, then there is hardly any user authentication mechanism (decoupled from SSL/TLS session establishment) that cannot be defeated or misused. This fact is usually neglected when people talk about the (perceived) security of SSL/TLS-based e-commerce applications, such as Internet banking or Internet voting.

In the literature, there is surprisingly little work on technologies and techniques to protect SSL/TLS-based e-commerce applications against MITM attacks. We believe that this topic is highly relevant and has not received adequate attention, either in the literature or in practice. In [OHB06], we therefore proposed *SSL/TLS session-aware user authentication* (TLS-SA) as a technological approach to fill this gap and to provide a lightweight alternative to the deployment and use of public key certificates on the client side. That paper presents a basic solution to implement TLS-SA, employing impersonal authentication tokens.

In this article, we put TLS-SA into perspective. More specifically, we take a closer look at MITM attacks in Section 2 and alternative countermeasures and their limitations in Section 3. In Sections 4 and 5 we present TLS-SA and introduce different extensions. In addition to summarizing the original, basic approach based on impersonal tokens, we show how to build implementations based on challenge-response systems and one-time passwords. Finally, we draw conclusions and provide an outlook in Section 6.

2 MITM Attacks

According to RFC 2828, an MITM attack refers to "a form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication association." Hence, the major characteristics of MITM attacks are (i) that they are active attacks, and (ii) that they target the associations between the communicating entities (rather than the entities or the communication channels between them). Note that in the literature, an MITM that carries out an active attack in real-time is sometimes called *adaptive*. We do not use this term and MITM attacks are adaptive by default.

In a typical setting, the MITM places himself between the user and the server in a way that he can talk to the user and the server separately, whereas the user and the server think that they are talking directly with each other. In an SSL/TLS setting, which is standard in Internet banking and other e-commerce applications, there are many possibilities for mounting an MITM attack. The user may be directed to the MITM using standard phishing techniques. Other possibilities are where the MITM employs Address Resolution Protocol (ARP) cache poisoning, or Domain Name System (DNS) spoofing (recently, the term "pharming" has been coined to refer to DNS spoofing attacks, such as local DNS cache poisoning).

The best way to think about an MITM attack is to consider an adversary that represents an SSL/TLS proxy server (or relay) between the user and the server. Neither the user nor the server are aware of the MITM. Cryptography makes no difference here as the MITM is in the loop and can decrypt and reencrypt all messages that are sent back and forth. Also, an MITM need not operate alone and there may be many entities involved in an MITM attack. One entity may be located between the client and the server, whereas the other entities may be located elsewhere. In this case, the corresponding attacks are known as *Mafia* or *terrorist* *fraud*, and the underlying problem is known as the *chess grandmaster problem*. For the purpose of this article, we do not care whether the adversary is acting as a single-entity or a multiple-entity MITM. Instead, we use the term MITM attack to refer to all types of attacks in which at least one entity is located between the client and the server.

MITM attacks can be devastating. If, for example, the user authenticates himself to an application server, then he reveals his credentials to the MITM and the MITM can misuse them to spoof the user. If the user employs an OTP system, then the MITM can grab the OTP (which is typically valid for at least a couple of seconds) and reuse it to spoof the user. If the user employs a C/R system, then again the MITM can simply send back and forth the challenge and response messages. Even if the user employed a zero-knowledge authentication protocol [FS87], the MITM would still be able to relay the messages and spoof the user accordingly. The zero-knowledge property does not, by itself, provide protection against MITM attacks—it only protects against information leakage related to the user's secret.

Given the above, it is perhaps not surprising that most currently deployed user authentication mechanisms fail to provide protection against MITM attacks, even when they run on top of the SSL/TLS protocol. We see two main problems responsible for this failure.

1. SSL/TLS server authentication is usually done poorly by the naive end user, if done at all.
2. SSL/TLS session establishment is usually decoupled from user authentication.

The first problem leads to a situation in which the user talks to the MITM, thereby revealing his credentials to the MITM. The second problem means that the credentials revealed by the user can be reused by the MITM to spoof the user to the origin server. To counter MITM attacks, it suffices to solve either of these problems. Solving the first requires hard-coded server certificates or dedicated client software, whereas the second requires modifications to SSL/TLS or to the authentication protocols used. As explained later in this article, we think that the second possibility is more appropriate for the Internet.

3 Countermeasures and Related Work

In spite of the fact that MITM attacks pose a serious threat to SSL/TLS-based e-commerce applications, there are only a few countermeasures and surprisingly little related work. Moreover, the situation is often misunderstood. For example, it is often stated within the security industry that strong (possibly two-factor) user authentication mechanisms are needed to thwart MITM attacks.[1] This statement is fundamentally wrong. Vulnerability to MITM attacks is not a user authentication problem—it is a server authentication problem. MITM attacks are possible because SSL/TLS server authentication is done poorly by the user (see the first point enumerated above). In other words, if users properly authenticated the server with which they establish an SSL/TLS session, then they would be protected against MITM attacks. Unfortunately, this is not the case and it is questionable whether it is possible at all. Note that an MITM can employ many tricks to give the user the impression of being connected to an origin server, for example, using visual spoofing, which is becoming increasingly popular. In the most extreme case, one may think of an MITM that is able to control the graphical user interface of the browser. Also, we have seen phishing sites using valid certificates. We refer to the case in which a phisher employed a valid certificate for www.mountain-america.com and www.mountain-america.net to spoof the Web site of the

[1]www.antiphishing.org/sponsors_technical_papers/PHISH_WP_0904

Mountain America Credit Union at www.mtnamerica.org. In such a setting, most users are unable to recognize that they are subject to an MITM attack.

Along the same line of argumentation, it is important to note that the SSL/TLS protocol has been designed to protect against MITM attacks. In addition to the requirement that certificate-based server authentication must be done properly, SSL/TLS-based MITM protection requires that all clients are equipped with public key certificates. This requirement could be relaxed, if one extended the SSL/TLS protocol with alternative client authentication methods (in addition to certificate-based authentication). In fact, there are efforts within the IETF to specify ciphersuites for the TLS protocol that support authentication based on pre-shared keys (PSKs) [ET+05, BH06]. Also, the adoption of password-based key exchange protocols was proposed in [S+01], and the use of the Secure Remote Password (SRP) is specified in an Internet-Draft. We believe that all of these efforts are important, but that there is still a long way to go until the corresponding TLS extensions are available, implemented, and deployed. In the meantime, one cannot count on the security properties of the SSL/TLS protocol alone. Instead, one must assume a setting in which the SSL/TLS protocol is only used to authenticate the server and the user authenticates himself on top of an established SSL/TLS session using some additional authentication mechanism.

In addition to the SSL/TLS protocol and extensions thereof, different cryptographic techniques have been proposed to protect users against MITM attacks. We list some of them here, in chronological order:

- Rivest and Shamir proposed the *Interlock* protocol [RS84] that was later shown to be vulnerable when used for authentication [BM94].
- Asokan et al. proposed protection mechanisms to secure tunneled authentication protocols against MITM attacks [ANN03] that are conceptually related to our proposal.
- Jakobsson and Myers proposed a technique called *delayed password disclosure* (DPD[2]) that can be used to complement a password-based authentication and key exchange protocol to protect (only) against a special form of MITM attack—the so-called *doppelganger window attack*.
- Kaliski and Nyström proposed the use of a *password protection module* (PPM[3]) to provide protection against MITM attacks. The PPM is a trusted piece of software that utilizes password hashing to generate a passcode that is unique for a user and an application in question.
- Parno et al. proposed a *Phoolproof* anti-phishing system that employs trusted devices, such as Bluetooth-enabled smartphones, to protect users against MITM attacks [PKP06].

In addition, as an alternative to employing cryptographic techniques and protocols, some researchers have suggested employing multiple communication channels and channel hopping to thwart MITM attacks (e.g., [ASS03]).

There is a steadily increasing number of e-commerce applications—especially in Europe—that authenticate users by sending short messaging system (SMS) messages that contain transaction authentication numbers (TANs) and require that users enter these TANs when they log in. Sending an SMS is an example of using two communication channels or two-factor authentication (the mobile phone being the second factor). While it has been argued that this mechanism protects against MITM attacks, unfortunately, this is not the

[2]http://www.informatics.indiana.edu/markus/stealth-attacks.htm
[3]http://www.rsasecurity.com/rsalabs/staff/bios/bkaliski/publications/other/kaliski-authentication-risk-readiness-bits-2004.ppt

case. If an MITM is located between the user and the server, then he need not eavesdrop on the SMS messages; all he needs to do to spoof the user is to forward the TAN submitted by the user on the SSL/TLS session. If one wants to work with TANs distributed via SMS messages, then one has to work with transaction-based TANs.[4] For each transaction submitted by the user, a summary is returned to the user together with a TAN in an SMS message. To confirm the transaction, the user must validate the content of the summary and, if correct, enter the corresponding TAN. The down-side of this proposal is that transaction-based TANs are expensive (perhaps prohibitively so), they are not particularly user-friendly, they do not protect the privacy of the user, and they are not even completely secure—an MITM can still attack the parts of a transaction that are not part of the summary.

We think that all technologies and techniques proposed so far either fail to adequately thwart MITM attacks or have severe disadvantages when it comes to a large-scale deployment. This leads us to the following proposal.

4 SSL/TLS Session-Aware User Authentication

Recall from the end of Section 2 that any effective countermeasure against MITM attacks in an SSL/TLS setting can either enforce proper server authentication or combine user authentication with SSL/TLS session establishment. We think that the first possibility asks too much of the user. The validation of public key certificates (including, for example, revocation checking) is too involved for the casual user. This is particularly true for deciding what to do when something illegitimate takes place. Users normally click through any dialogue that pops up to warn them. We therefore focus on the second approach and examine possibilities for combining user authentication with SSL/TLS session establishment, which we refer to as *SSL/TLS session-aware user authentication* (TLS-SA).

The main idea behind TLS-SA is to make the user authentication depend not only on the user's (secret) credentials, but also on state information related to the SSL/TLS session in which the credentials are transferred to the server. The rationale is that the server should have the possibility of determining whether the SSL/TLS session in which it receives the credentials is actually the same as the one the user employed when he sent out the credentials in the first place. In typical secure online applications, server operators such as banks typically are much better equipped to master complex tasks such as validating cryptographic protocol outcomes. The server is normally run in a data-center managed by sophisticated administrators while the clients, which are increasingly small mobile devices, are used by comparatively unsophisticated users.

- If the two sessions are the same, then there is probably no MITM involved.
- If the two sessions are different, then something abnormal is taking place. It is then likely that an MITM is located somewhere between the user's client system and the server.

Using TLS-SA, the user authenticates himself by providing a user authentication code (UAC) that depends on both the credentials and the SSL/TLS session (in particular, on information from the SSL/TLS session state that is cryptographically hard to alter). An MITM who gets hold of the UAC can no longer misuse it by simply retransmitting it. The key point is that the UAC is bound to a particular SSL/TLS session. Hence, if the UAC is submitted on a different session, then the server can detect this fact and drop the session accordingly. This is illustrated in Figure 1. There is a blue SSL/TLS session between the user and the MITM and a red SSL/TLS session between the MITM and the server. The UAC is bound to the blue

[4]http://www.cryptomathic.com/pdf/The%20Future%20of%20Phishing.pdf

session. So, if the MITM comes along and submits the UAC on the red session, then the server will detect the misuse and drop the session.

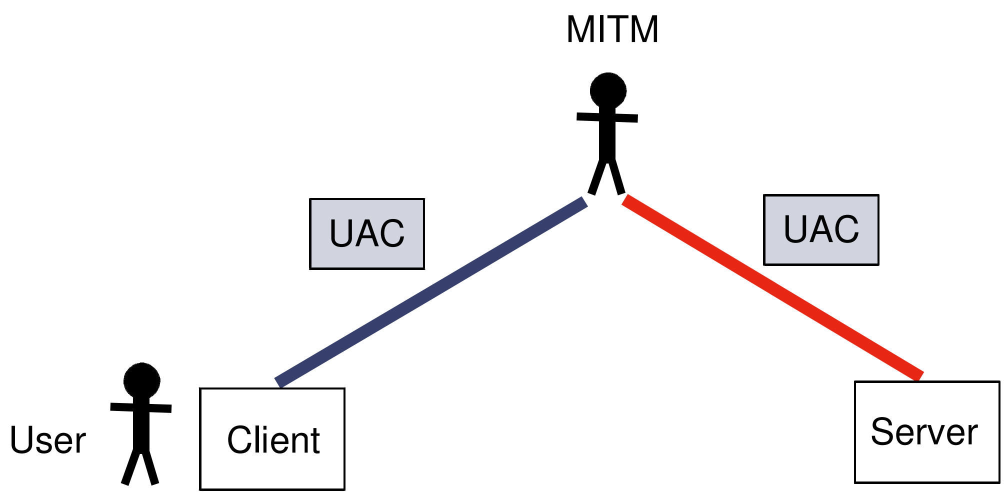

Figure 1 An MITM trying to submit the user credentials (i.e., the UAC) on a “wrong” SSL/TLS session

If a user wants to protect himself against a server not dropping the session before he or she reveals sensitive information (e.g., credit card information), then he or she can request a server authentication code (SAC) for mutual authentication. The generation and verification of a SAC is analog to a UAC and not further addressed in this overview article.

5 Implementing TLS-SA

There are many possibilities to implement TLS-SA. Note that TLS-SA is not a user authentication mechanism or system per se. Rather, it is a technological approach that can be used to make a given authentication mechanism or system be SSL/TLS session-aware and hence resistant against MITM attacks. The way this is done depends on the actual authentication mechanism or system in use. If, for example, users are authenticated with PINs, then there is a basic solution proposed in [OHB06] that employs impersonal authentication tokens. We introduce this solution first, before we expand on possibilities for making C/R and OTP systems SSL/TLS session-aware.

5.1 Basic Solution

In our basic solution, a user U is equipped with an impersonal authentication token T with a small display. U employs a client (i.e., browser) C to access an SSL/TLS-based application on server S. U is identified with ID_U and holds PIN_U (i.e., a PIN shared with S). T is identified by a serial number SN_T that may, for example, be imprinted on the back side of the token. Furthermore, T is equipped with both a public key pair (k,k^{-1}) and a secret token key K_T shared with S. The keys k and k^{-1} are the same for all tokens, which is why the tokens are impersonal, whereas K_T is unique and specific to T. Note, however, that K_T is not specific to the user, and hence the tokens need not be personalized (this is the main advantage of using impersonal tokens in the first place). K_T can be generated randomly, or pseudo-randomly, using a master key MK, i.e., a key held exclusively by S:

$$K_T = E_{MK}(SN_T)$$

In the first case, where K_T is generated randomly, all token keys must be stored on the server side. However, in the second case, where K_T is derived from SN_T, there is no need to centrally store all token keys. Instead, K_T can be generated dynamically from SN_T and MK.

In order to access S, U directs C to S. C and S then try to establish an SSL/TLS session using the SSL/TLS handshake protocol. As part of this protocol, S authenticates itself using a public key certificate. S is configured in a way that it always requires certificate-based client authentication by sending an SSL/TLS CertificateRequest message to C. When C receives this message, it knows that it must authenticate itself by returning a Certificate message and a properly signed CertificateVerify message to S. The CertificateVerify message comprises a digitally signed hash value *Hash* of all messages previously exchanged during the execution of the SSL/TLS handshake protocol. Part of these messages is the server's Certificate message, which comprises the server's public key certificate (and hence the server's public key, included in this certificate). Consequently, the CertificateVerify message is logically bound to the server's public key.

The digital signature is generated by T using its private key k^{-1}. Due to the fact that T is an impersonal token, the CertificateVerify message neither authenticates the token nor the client. It only ensures that C uses a token to establish an SSL/TLS session with S and that the token has access to *Hash*. Said another way, the token represents a trusted observer for *Hash* and for enforcing a proper execution of TLS-SA. In addition to providing a properly signed CertificateVerify message to S, T also renders a shortened version of

$$N_T = E_{KT}(\mathit{Hash}) \tag{1}$$

on its display (for example, in decimal notation). In this case, E represents an encryption function that is keyed with K_T. Alternatively, N_T could represent a message authentication code (MAC) computed with a one-way hash function keyed with K_T.[5] For example, the HMAC construction (cf. RFC 2104) can be used to generate

$$N_T = HMAC_{KT}(\mathit{Hash}). \tag{2}$$

In either case, N_T can be shortened to the length of PIN_U by truncating it. This value must then be combined with PIN_U to generate a UAC that is valid for exactly one SSL/TLS session initiated by U. If *f* represents such a combination function, then the UAC can be expressed as

$$UAC = f(N_T, PIN_U) \tag{3}$$

There are many ways to define an appropriate function *f*. One possibility is digit-wise addition, modulo 10. In this case, the UAC is the digit-wise sum, modulo 10, of N_T and PIN_U, which is a function simple enough for users to compute themselves (in their heads). Another possibility is to use PIN_U to select specific digits of N_T. This construction is, for example, employed by PINsafe of Swivel (in Swivel's terminology, PIN_U refers to the user PIN, N_T refers to the security string, and the UAC refers to the one-time code). If the token comprises a keypad for entering PIN_U, then the token can also be used to compute *f*. It goes without saying that *f* can then be more complex.

[5]In some circumstances, the use of a MAC is advantageous because keyed one-way hash functions are generally more efficient than symmetric encryption systems and because there are usually no regulations restricting the use of (keyed) one-way hash functions, in contrast to the use of encryption systems.

After a server-authenticated SSL/TLS session is successfully established between C and S, S can authenticate U by requesting ID_U, SN_T, and a valid UAC for the SSL/TLS session in use. Note that the user need not enter SN_T if this value is included in the public key certificate for T's private key k^{-1}. In this case, S can retrieve SN_T from the certificate, but the certificate is then token-specific. Alternatively, one could also provide for the user to temporarily register with a specific token. In this case, S can retrieve ID_U from its registration database and set it as a default value in the user authentication process. Note, however, that the binding between SN_T and ID_U is still weak. On the server side, S can verify the UAC because it knows *f* and PIN_U, and because it can reconstruct N_T (since it knows *Hash* and MK to generate K_T).

In theory, it is feasible to transfer the UAC as part of the SSL/TLS CertificateVerify message so that the user authentication can be handled entirely by the token (and hence the user does not have to enter anything in a Web form). For example, T can digitally sign both the *Hash* value and the UAC. Alternatively, T can digitally sign a keyed hash value (instead of the hash value *Hash*), where the UAC represents the key, *Hash* represents the argument, and the HMAC construction can be used to key the hash function. Last but not least, T can only send the keyed hash value (instead of the digital signature). Because all of these possibilities require changes in the SSL/TLS handshake protocol, they are not further addressed in this article.

5.2 C/R Systems

A C/R system is an authentication system where the entity that is authenticated (typically the client) is provided with a challenge that it must return an appropriate response to the entity that is authenticating it (typically the server). C/R systems are often implemented in hardware and the corresponding (hardware) tokens may be connected to the client systems using some cryptographic token interface standard, such as PKCS #11 or Microsoft's Cryptographic API (CAPI).

On the conceptual level, there is a simple and straightforward possibility to make a C/R system SSL/TLS session-aware: instead of having the server provide a challenge to the client, the client and the server both employ a value derived from the SSL/TLS session state as a challenge. We distinguish between two cases depending on whether or not the token is connected to the client system.

1. If the token is connected, then *Hash* can be sent to the token and the token can compute N_T according to formula (1) or (2). N_T can either be displayed so the user can compute the UAC or the user must additionally input PIN_U into the token, so that the token can compute the UAC (this is the preferred choice). The UAC can then be manually copied by the user from the token display to the appropriate Web form.
2. If the token is not connected, then the situation is somewhat more involved. In this case, there is no communication path between the client and the token and hence there must be another possibility for communicating SSL/TLS state information from the browser to the token (see below).

The main advantages of the first case, where the token is connected, are that the user interaction is simple and that one can use N_T and possibly the UAC in its entire length, thereby providing better security and relaxing the requirements on what parts of the user's screen must not be controlled by the attacker. The main disadvantage is the necessity of having a free port to connect the device to and the need to install a token driver on the client system (unless the operating system has a standard device driver available such as currently, for example, in Windows XP-based client systems with Microsoft Internet Explorer). Also, connectivity typically adds to the cost of the token hardware.

The main advantage of the second case, where tokens are not connected, is that there is no need for a physical token interface and hence a token driver need not be installed. The main disadvantage is that another possibility to communicate SSL/TLS state information from the browser to the token is required. One possibility is to have the browser display *Hash*, which the user must then enter into the token. This has the disadvantage that *Hash* is quite long (i.e., 36 bytes), and hence it is unreasonable to ask the user to manually enter it into the token. An alternative is the use of an optical transmission channel, such as the flickering code employed by AXSionics.[6] Another alternative is the use of a special function to compress or truncate *Hash* to only a few bits. This alternative is explored next.

Compressing *Hash* is tricky and care is required at this step. If one works with a deterministically compressed or truncated *Hash* value, then an MITM can still go for colliding *Hash* values. One possibility to protect against this attack is to have the browser pseudo-randomly select and work with a subset of the *Hash*'s bits to form the challenge. More specifically, the browser pseudo-randomly selects a few position indices and bit sequences that begin at these positions, and concatenates them to form a challenge. The number and lengths of the bit sequences depend on the maximum length of the challenge. The challenge can be displayed by the browser and the user can enter it into his token. The token, in turn, can then generate a response. If the token implements a block cipher and holds a token key K_T, then this key can be used to encrypt the challenge according to

$$\mathit{Response} = E_{KT}(\mathit{Challenge} || \mathit{Padding}) \qquad (4)$$

Note that *Challenge* is typically shorter than the block length of the block cipher and hence one must introduce some padding. Also note that the block length is important because one must avoid the case in which the output of the block cipher is too long (since the user must manually enter this value into a Web form). In a typical setting, *Challenge* may comprise 4 to 8 decimal characters, whereas *Response* may comprise up to 13 alphanumerical characters (e.g., uppercase letters and decimal characters). If $l=l_{Challenge}$ is the maximum length of the challenge (in decimal digits), then one has $b = \lfloor \log_2(10^l-1) \rfloor$ bits to represent the challenge. For $l_{Challenge}=4$, this corresponds to b=13, and for $l_{Challenge}=8$, this corresponds to b=26. So if the challenge is 8 decimal digits long, then one can employ 26 bits to encode the challenge. In equation (4) above, the response refers to one ciphertext block. This means that the length of the response is equal to the block length of the cipher in use. For DES and 3DES, for example, this is 64 bits. If we employ the Base-32 encoding scheme that comprises 32 characters encoded in 5 bits each, then we need $\lceil 64/5 \rceil=13$ alphanumeric characters to encode the response. If, however, we employ the Base-64 encoding scheme that comprises 64 characters encoded in 6 bits each, then we can reduce the length of the response to the more realistic size of $\lceil 64/5 \rceil=11$ characters.

We have built a prototype implementation of TLS-SA that employs C/R tokens, described in detail in [O+07]. On the client side, the implementation employs non-connected C/R tokens from Vasco and a specially crafted plugin for Microsoft Internet Explorer. On the server side, the implementation employs a lightly modified web server. Our implementation both demonstrates the technical feasibility of TLS-SA and provides a testbed for further analysis and improvement.

[6]http://www.axsionics.com

5.3 OTP Systems

In contrast to C/R systems, OTP systems do not work with challenges. Instead, the entity that is authenticated provides an OTP to the entity that is authenticating it. No interaction or handshake usually takes place between the client and the server. Roughly speaking, there are three classes of OTP systems.

1. *Physical lists of OTPs:* Examples include scratch lists and access cards, as well as lists of TANs and indexed TANs (iTANs).
2. *Software-based OTP systems:* Examples include Lamport-style OTP systems, such as Bellcore's S/Key and the one-time passwords in everything (OPIE) system.
3. *Hardware-based OTP systems:* Examples include SecurID and SecOVID tokens. Note that most hardware-based OTP systems are not connected to the client systems. This simplifies the deployment of the tokens and makes them resistant to many malware attacks.

In [OHB06], we indicated how to use impersonal authentication tokens to complement hardware-based OTP systems in the sense that the resulting (combined) authentication system is SSL/TLS session-aware. The idea is that U employs the OTP as input for *f* (instead of PIN_U) in formula (3). Hence, the UAC computed is

$$UAC = f(N_T, OTP). \quad (3)$$

Everything else (including the construction of N_T) remains unchanged. The disadvantage of this approach is that the user must have two tokens (i.e., the original OTP token and the impersonal authentication token) or the tokens must be integrated into one. The first possibility is likely to be unacceptable in practice, whereas the second possibility will take some time until integrated tokens appear on the marketplace. Thus, the use of software-based OTP systems seems to be more appropriate.

For the classes of OTP systems listed above, the first class is definitively the most difficult one to make SSL/TLS session-aware. If we only have a physical (paper) list of OTPs, then a technique similar to the one to make C/R systems be SSL/TLS session-aware may be employed. If a SSL/TLS session between the browser and the server is established, then the user selects the next OTP not yet used and enters it into the browser. The browser, in turn, interprets this value as an index into the *Hash* value. A specific number of bits is then extracted from the *Hash* value and this bit sequence represents the UAC. The browser may display the UAC and the user may enter it in a Web form (instead of the OTP).

6 Conclusions and Outlook

Many SSL/TLS-based e-commerce applications employ traditional authentication mechanisms on the client side. It is not widely known, even within the security community, that most of these mechanisms—if decoupled from SSL/TLS session establishment—are vulnerable to MITM attacks. Few institutions have fully recognized the seriousness of this threat and of those who have, many have declared client certificate authentication as the "strategic solution." To date, the often discussed obstacles to a full-PKI roll-out have prevented the successful, large-scale deployment of this solution. There does not exist a generally acceptable PKI-based solution that provides the degrees of freedom needed and supports the variety of platforms required by customers to use services like Internet banking.

These problems motivated our proposal in [OHB06] of TLS-SA based on impersonal authentication tokens. In this article, we put TLS-SA into perspective. More specifically, we took a closer look at MITM attacks, countermeasures and related work, TLS-SA, and—most importantly—possibilities to implement it in different real-world settings. As mentioned in the introduction, we have tested some of our ideas by building a proof of concept implementation of TLS-SA, employing C/R tokens. More recently, we have started work on making smartcards that are SSL/TLS session-aware and that conform to Eurocard, MasterCard, and Visa's (EMV) Chip Authentication Program (CAP). This is important as it is likely that EMV-CAP debit cards will be widely deployed (at least in Europe) and will, for example, also be used for Internet banking.

Overall, TLS-SA is a promising approach to solving the MITM problem. It leverages the legacy authentication mechanisms and systems that the broad masses have become accustomed to using. It neither requires a full-fledged PKI, as is mandatory when performing authentication using client-side certificates, nor does it unambiguously identify the parties involved, also towards a illegitimately observing third party. Hence, we firmly believe that TLS-SA provides as a lightweight and privacy-enhancing alternative to the deployment and use of client-side certificates.

References

[ANN03] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols," *Proceedings of the International Workshop on Security Protocols*, 2003, pp. 15–24 (also available as IACR ePrint 2002/163).

[ASS03] Alkassar, A., Stüble, C., and A.-R. Sadeghi, "Secure Object Identification—or: Solving The Chess Grandmaster Problem," *Proceedings of the 2003 Workshop on New Security Paradigms*, Ascona, Switzerland, ACM Press, NY, 2003, pp. 77–85.

[BH06] Badra, M., and I. Hajjeh, "Key-Exchange Authentication Using Shared Secrets," *Computer*, Vol. 39, Issue 3, March 2006, pp. 58–66.

[BM94] Bellovin, S.M., and M. Merritt, "An Attack on the Interlock Protocol When Used for Authentication," *IEEE Transactions on Information Theory*, Vol. 40, No. 1, January 1994.

[DR06] Dierks, T., and E. Rescorla, "The TLS Protocol Version 1.1," RFC 4346, April 2006.

[ET+05] Eronen, P., and H. Tschofenig (Eds.), "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)," Standards Track RFC 4279, December 2005.

[FS87] Fiat, A., and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," *Proceedings of CRYPTO '86*, Springer, LNCS 263, 1987, pp. 186–194.

[O+07] Oppliger, R., et al., "A Proof of concept Implementation of SSL/TLS Session-Aware User Authentication," *Proceedings of Kommunikation in Verteilten Systemen (KiVS 2007)*, Springer-Verlag, 2007, pp. 225–236.

[OHB06] Oppliger, R., Hauser, R., and D. Basin, "SSL/TLS Session-Aware User Authentication—Or How to Effectively Thwart the Man-in-the-Middle," *Computer Communications*, Vol. 29, Issue 12, August 2006, pp. 2238–2246.

[PKP06] Parno, B., Kuo, C., and A. Perrig, "Phoolproof Phishing Prevention," *Proceedings of Financial Cryptography and Data Security*, Springer-Verlag, 2006

[RS84] Rivest, R.L., and A. Shamir, "How to Expose an Eavesdropper," *Communications of the ACM*, Vol. 27, No. 4, 1984, pp. 393–395.

[S+01] Steiner, M., et al., "Secure Password-Based Cipher Suite for TLS," *ACM Transactions on Information and System Security (TISSEC)*, Vol. 4, No. 2, May 2001, pp. 134–157.