# SSL/TLS Session-Aware User Authentication

*Rolf Oppliger,* eSECURITY Technologies
*Ralf Hauser,* PrivaSphere AG
*David Basin,* ETH Zurich

**Overall, transport-layer security with session-aware user authentication offers a promising approach to solving man-in-the-middle attack problems by leveraging the legacy authentication mechanisms and systems that the general public has become accustomed to using.**

Most e-commerce applications used today employ the Secure Sockets Layer (SSL) or Transport Layer Security (TLS)[1] protocol to authenticate the server and cryptographically protect the communication channel between the client and server. Although SSL/TLS provides support for user authentication based on public key certificates, in practice, because of 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, and strong authentication mechanisms such as one-time password (OTP) or challenge-response (C/R) systems.

While developers consider the SSL and TLS protocols to be sound and secure in practice, the vast majority of SSL/TLS-based e-commerce applications that employ 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, 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, most user authentication mechanisms (decoupled from SSL/TLS session establishment) can be defeated or misused. This is usually neglected when people talk about the perceived security of SSL/TLS-based e-commerce applications, such as Internet banking or remote Internet voting.

The literature reveals surprisingly little work on technologies and techniques to protect SSL/TLS-based e-commerce applications against MITM attacks. SSL/TLS session-aware user authentication (TLS-SA) has been proposed as a technological approach to fill this gap and provide a lightweight alternative to the deployment and use of public-key certificates on the client side.[2] The original approach comes along with a basic solution to implement TLS-SA, employing impersonal authentication tokens.

## 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 to masquerade as one or more of the entities involved in a communication association." Hence, MITM attacks represent active assaults that target the associations between the communicating entities rather than the entities or communication channels between them. In the literature, an MITM that executes an active attack in real time is called *adaptive*. We do not use this term because MITM attacks are adaptive by default.

In a typical setting, the MITM places himself between the user and the server so that he can talk to the user and server separately, whereas the user and server think 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


Published by the IEEE Computer Society
0018-9162/08/$25.00 © 2008 IEEE

mounting an MITM attack. For example, the user might be directed to the MITM using standard phishing techniques. Other possibilities include the MITM employing address resolution protocol (ARP) cache poisoning or domain name system (DNS) spoofing. The recently coined term *pharming* refers to DNS spoofing attacks, such as local DNS cache poisoning.

Think of an MITM attack as an adversary that represents an SSL/TLS proxy server, or relay, between user and server. Neither the user nor the server is aware of the MITM. Cryptography makes no difference here as the MITM is in the loop and can decrypt and reencrypt all messages sent back and forth. Also, an MITM need not operate alone—there could be many entities involved in an MITM attack. One entity might be located between the client and server, whereas the other entities might be located elsewhere. In this case, the corresponding attacks are sometimes called Mafia or terrorist fraud attacks and the underlying problem dubbed the *chess grandmaster problem*. We use the term MITM attack here to refer to all types of attacks in which at least one entity is located between the client and server. If the adversary has compromised the client software, people sometimes use the analogous term *man-in-the-browser* (MITB). Such attacks are powerful and difficult to protect against.

MITM attacks can be devastating. If, for example, the user authenticates himself to an application server and then reveals his credentials to the MITM, the MITM can misuse them to spoof the user. If the user employs an OTP system, 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, again the MITM can simply send the challenge and response messages back and forth. Even if the user employs a zero-knowledge authentication protocol,[3] 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.

Thus, 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. Two main problems are responsible for this failure: the naïve end user usually does SSL/TLS server authentication poorly if at all, and developers usually decouple SSL/TLS session establishment from user authentication. The first problem leads to a situation in which users talk to the MITM, thereby revealing their credentials to him. The second problem enables the MITM to reuse the credentials revealed by users to spoof them.

To counter MITM attacks, solving either of these problems suffices. Solving the first problem requires hard-coded server certificates or dedicated client software, whereas fixing the second problem requires modifications to SSL/TLS or the authentication protocols used. We consider the second solution more appropriate for the Internet, mainly because the overall modifications are smaller.

## COUNTERMEASURES AND RELATED WORK

Even though MITM attacks pose a serious threat to SSL/TLS-based e-commerce applications, researchers have devised only a few countermeasures and have undertaken surprisingly little related work. One reason for this may be that the situation is often misunderstood. For example, analysts within the security industry often state that only strong, possibly two-factor, user authentication mechanisms can thwart MITM attacks (such as www.antiphishing.org/sponsorstechnicalpapers/PHISHWP0904). This statement is flawed. Vulnerability to MITM attacks is not a *user* authentication problem but rather a *server* authentication problem. MITM attacks succeed because the user executes SSL/TLS server authentication poorly. If users properly authenticated the server with which they establish an SSL/TLS session, they would be protected against MITM attacks. Unfortunately, this is not the case and might not be possible at all.

**It is perhaps not surprising that most currently deployed user authentication mechanisms fail to provide protection against MITM attacks.**

An MITM can employ many tricks to give the user the impression of being connected to the intended website server—by using the increasingly popular technique of visual spoofing, for example. In the most extreme case, consider an MITM that can control the browser's graphical user interface. Also, we have seen phishing sites that use valid certificates. For example, a phisher employed a valid certificate for www.mountain-america.com and www.mountain-america.net to spoof the website of the Mountain America Credit Union (www.mtnamerica.org). In such a setting, most users are powerless to recognize they are experiencing an MITM attack.

Along these same lines, 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 be equipped with public-key certificates. This requirement could be relaxed if we extended the SSL/TLS protocol with alternative client authentication methods. Indeed, efforts have been promoted within the IETF to specify ciphersuites for the TLS protocol that support authentication based on preshared keys (PSKs).[4,5] Also, some researchers[6] have proposed the adoption of password-based key-exchange protocols, while the use of the Secure Remote Password

(SRP) protocol is specified in an Internet draft (www.ietf.org/mail-archive/web/i-d-announce/current/msg15532.html).

In addition to the SSL/TLS protocol and the extensions thereof, a few cryptographic techniques aimed at protecting users against MITM attacks can also be employed. In chronological order they are as follows:

- Ron Rivest and Adi Shamir proposed the Interlock protocol,[7] later shown to be vulnerable when used for authentication.[8] Protection against this vulnerability requires turning the simple protocol into a complex multiple-round protocol.
- N. Asokan and colleagues proposed protection mechanisms to secure tunneled authentication protocols against MITM attacks[9] that are conceptually related to our proposal.
- Markus Jakobsson and Steve Myers proposed a *delayed password disclosure* technique (www.informatics.indiana.edu/markus/stealth-attacks.htm) that can be used to complement a password-based authentication and key exchange protocol to protect against a special form of MITM attack—the so-called doppelganger window attack.
- Burt Kaliski and Magnus Nyström proposed using a *password protection module* (PPM) to provide protection against MITM attacks.[10] The PPM is a trusted piece of software that uses password hashing to generate a passcode unique for a given user and application.
- Bryan Parno and colleagues proposed the *Phoolproof antiphishing system* that employs trusted devices, such as Bluetooth-enabled smart phones, to protect users against MITM attacks.[11]

Instead of cryptographic techniques and protocols, some researchers have suggested combining multiple communication channels with channel hopping to thwart MITM attacks.[12]

A steadily increasing number of e-commerce applications—especially in Europe—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 message 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, this is, unfortunately, not the case. If an attacker positions an MITM between the user and server, he need not eavesdrop on the SMS messages; to spoof the user, he can forward the TAN submitted by the user on the SSL/TLS session.

Distributing TANs via SMS messages would require working with transaction-based TANs—a technique commonly referred to as *transaction signing*. In an Internet banking environment, for each transaction that a user submits, the server returns a summary, together with a TAN, in a SMS message. To confirm the transaction, the user must enter the corresponding TAN, but only if the transaction summary is correct. The drawbacks of this proposal are that transaction-based TANs are expensive, they are not particularly user-friendly, they do not protect the user's privacy on the primary channel over the Internet, and they are not even completely secure. For example, they may expose customer-related data on the secondary channel that may not be properly secured. Anyway, an MITM can still attack the parts of the transaction that are separate from the summary. Transaction signing and server-side fraud detection (using, for example, expert systems) are about to become standard measures to protect Internet banking infrastructures against direct fraud. They do not, however, improve the security or privacy of an SSL/TLS session per se.

**All the technologies and techniques proposed so far either fail to adequately thwart MITM attacks or have severe disadvantages when it comes to large-scale deployment.**

Thus, all the technologies and techniques proposed so far either fail to adequately thwart MITM attacks or have severe disadvantages when it comes to large-scale deployment.

## SSL/TLS SESSION-AWARE USER AUTHENTICATION

Any effective countermeasure against MITM attacks in an SSL/TLS setting must either enforce proper server authentication or combine user authentication with SSL/TLS session establishment. With respect to the first possibility, we think it asks too much of the user. The validation of public-key certificates (including, for example, revocation checking) is too involved. This is particularly true when deciding what to do when something illegitimate is taking 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. We use the term SSL/TLS session-aware user authentication (TLS-SA) to refer to this.

TLS-SA focuses mainly on making 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. It is based on the idea that the server should have the possibility of determining whether the SSL/TLS session in which it receives the credentials actually matches the

one the user employed when first sending out the credentials. Thus, if the two sessions match, it is very unlikely that an MITM is involved, but if they differ, something abnormal must be taking place. Most likely, an MITM has been positioned 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 his credentials and the SSL/TLS session, in particular, on information from the SSL/TLS session state that is cryptographically hard to alter. An MITM that 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, the server can detect this and drop the session accordingly, as Figure 1 shows. The figure depicts 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 session, so if the MITM submits the UAC on the red session, the server will detect the misuse and drop the session.

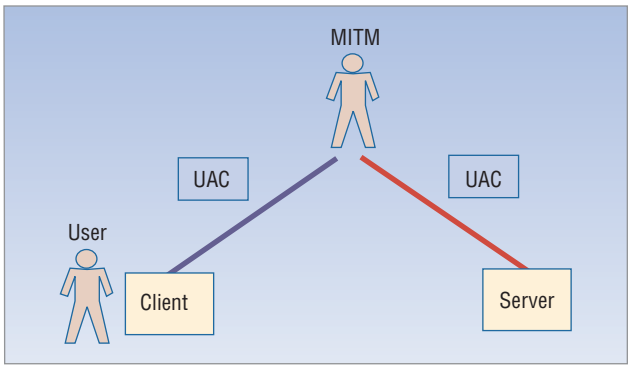


***Figure 1. Man-in-the-middle attack. This MITM seeks to submit the user authentication code (UAC) credentials on a "wrong" SSL/TLS session.***

## IMPLEMENTING TLS-SA

Although TLS-SA is not a user authentication mechanism or system per se, it can be implemented in many different ways. Using this technological approach can make a given authentication mechanism or system SSL/TLS session-aware and hence resistant to MITM attacks. How this occurs depends on the actual authentication mechanism or system used. If, for example, the system uses PINs to authenticate users, one proposed basic solution employs impersonal authentication tokens.[2]

### Basic solution

In the basic solution, a user $U$ is equipped with an authentication token $T$ with a small display. $U$ employs a client browser $C$ to access an SSL/TLS-based application on server $S$. $U$ is identified with $ID_U$ and holds $PIN_U$, a PIN shared with $S$. $T$ is identified by a serial number, $SN_T$, that might, for example, be imprinted on the back side of the token. Further, $T$ 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 (and hence the token is impersonal), whereas $K_T$ is unique and specific to $T$. Note, however, that $K_T$ is not specific to the user, hence the tokens need not be personalized, which is the main advantage of using impersonal tokens. $K_T$ can be generated randomly, or pseudorandomly, using a master key $MK$, typically held exclusively by $S$:

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

In the first case, where $K_T$ is generated randomly, the system must store all token keys on the server side. In the second case, however, where $K_T$ is derived from $SN_T$ the system does not need to store token keys centrally. Instead, the user can generate $K_T$ dynamically from $SN_T$ and $MK$.

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 so that it always requires certificate-based client authentication by sending an SSL/TLS `CertificateRequest` message to $C$. When $C$ receives this message, it knows it must authenticate itself by returning a `Certificate` 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. The server's `Certificate` message constitutes part of these messages, comprising the server's public-key certificate and hence the server's public key, included in this certificate. The `CertificateVerify` message is thus logically bound to the server's public key.

$T$ generates the digital signature using its private key $k^{-1}$. Given that $T$ is an impersonal token, the `CertificateVerify` message authenticates neither 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*. The token represents a trusted observer for *Hash* and for enforcing a proper execution of the TLS-SA. In addition to providing a properly signed `CertificateVerify` message to $S$, $T$ also renders a shortened version of

$$N_T = E_{K_T}(Hash) \quad (1)$$

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

$$N_T = HMAC_{K_T}(Hash). \quad (2)$$

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* valid for exactly one SSL/TLS session initiated by *U*. If *f* represents such a combination function, the UAC can be expressed as

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

There are many ways to define an appropriate function *f*, including, for example, digitwise addition modulo 10. In this case, the UAC is the digitwise sum, modulo 10, of $N_T$ and $PIN_U$, which is a function simple enough for users to compute on their own. Another possibility involves using $PIN_U$ to select specific digits of $N_T$, a construction that Swivel's PINsafe employs. In Swivel's terminology, $PIN_U$ refers to the user PIN, $N_T$ refers to the security string, and UAC refers to the one-time code. If the token possesses a keypad for entering $PIN_U$, the system also can use the token to compute *f*, and *f* can then be more complex.

After the system successfully establishes a server-authenticated SSL/TLS session 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. The user need not enter $SN_T$ if the public-key certificate for *T*'s private key $k^{-1}$ includes this value. In this case, *S* can retrieve $SN_T$ from the certificate, but the certificate then becomes token-specific. Alternatively, the system could also let the user 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 $ID_U$ remains 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 the *Hash* and *MK* values to generate $K_T$.

In theory, the token can transfer the UAC as part of the SSL/TLS `CertificateVerify` message so that the token can entirely handle the user authentication, thus removing the need for the user 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 system can use the HMAC construction to key the hash function. Last but not least, *T* can only send the keyed hash value instead of the digital signature.

## C/R systems

A C/R system authenticates an entity—typically the client—when it receives a challenge for which it must return an appropriate response to the entity authenticating it, typically the server. C/R systems are often implemented in hardware, and the system can connect the corresponding hardware tokens to the client systems using a cryptographic token interface standard, such as PKCS #11 or Microsoft's Cryptographic API (CAPI).

On the conceptual level, a simple and straightforward approach can be used to make a C/R system SSL/TLS session-aware, as follows: Instead of having the server provide a challenge to the client, the client and 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 connects to the client system.

1. If the token is connected, the system can send *Hash* to the token, and the token can compute $N_T$ according to formula (1) or (2). $N_T$ can either be displayed so that the user can compute the UAC, or—and this is the preferred choice—the user must additionally input $PIN_U$ into the token so that it can compute the UAC. The user can then manually copy the UAC from the token display to a Web form.
2. If the token is not connected, the situation becomes somewhat more involved. In this case, no communication path lies between the client and token, and thus the system must find another possibility for communicating SSL/TLS state information from the browser to the token. Ideally, when clicking on the symbol or icon to show the server certificate in a browser, it would also display SSL/TLS state information in a human-copyable form.

The token-connected first case offers the main advantages of keeping the user interaction simple and allowing use of $N_T$, and possibly the UAC, in its entire length, thereby providing better security. 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 that currently available in Windows-based client systems with Microsoft Internet Explorer. Further, enabling such connectivity typically adds to the cost of the token's hardware.

The second case offers the main advantage of not requiring that tokens be connected, thus removing the need for a physical token interface and driver installation. This case suffers from the disadvantage that it requires supplying another means for communicating SSL/TLS state information from the browser to the token.

One possibility is for the browser to display *Hash*, which the user must then enter into the token. This has the disadvantage that it asks a lot of the user to manually enter 36 bytes of information (the *Hash* length) into the token. The use of a flickering code, such as the one pioneered by AXSionics (www.axsionics.com), offers one alternative. Another alternative uses a special function to compress or truncate *Hash* to only a few bits. Compressing *Hash* is tricky and requires some care. Working with a deterministically compressed or truncated *Hash* value, however, can leave the system vulnerable to the following MITM attack: The MITM waits for a client system to establish

the first SSL/TLS session, then sets up a second SSL/TLS session to the origin server and modifies the `ServerHello` message to return in the first session such that the compressed or truncated *Hash* values of both sessions are the same. The MITM thus looks for a second-preimage of the *Hash* value for the compression or truncation function in use. Since this value determines the challenge, the MITM thus can simply forward the user's response to authenticate to the origin server.

Given the feasibility of such a challenge-collision attack, developers must be careful to properly compress or truncate the *Hash* value. For example, instead of working with the entire value, the browser can pseudorandomly select and work with a subset of its bits to form the challenge. More specifically, the browser can pseudorandomly select a few position indices and bit sequences that begin at these positions and concatenate them to form a challenge. The number and lengths of the bit sequences depend on the challenge's maximum length. The browser can display the challenge, and the user can enter it into his token. The token, in turn, can generate a response. If the token implements a block cipher and holds a token key $K_T$, the system can use this key to encrypt the challenge according to

$$Response = E_{K_T}(Challenge \,||\, Padding). \quad (4)$$

Because *Challenge* typically is shorter than the block cipher's block length, this option requires adding padding. Also the block length is important because developers must avoid the case where the block cipher's output is too long since the user must manually enter the value into a Web form. According to Equation 4, the response refers to one ciphertext block. This means that the response's length equals the block length of the cipher in use. For DES and 3DES, for example, this value would be 64 bits. If we employ the Base-32 encoding scheme that comprises 32 characters encoded in 5 bits each, we need $\lceil 64/5 \rceil = 13$ characters to encode the response. If, however, we employ the Base-64 encoding scheme that comprises 64 characters encoded in 6 bits each, we can reduce the response's length to the more realistic size of $\lceil 64/6 \rceil = 11$ characters. In either case, connectivity provides security and usability advantages compared to C/R tokens. One option would be to design C/R tokens that are unconnected by default, but which can be upgraded to provide better security and usability.

We have built a prototype implementation of TLS-SA that employs C/R tokens.[13] On the client side, the implementation employs nonconnected C/R tokens from Vasco and a specially crafted plug-in for Microsoft Internet Explorer. On the server side, the implementation employs a slightly modified Web server. Our implementation both demonstrates the technical feasibility of TLS-SA and provides a testbed for further analysis and improvement. For example, we are attempting to optimize resistance against challenge-collision attacks.[14]

**Our implementation both demonstrates the technical feasibility of TLS-SA and provides a testbed for further analysis and improvement.**

## OTP systems

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

- *Physical OTP lists*. Examples include scratch lists and access cards, as well as lists of TANs and indexed TANs (iTANs).
- *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.
- *Hardware-based OTP systems*. Examples include SecurID and SecOVID tokens.

Most hardware-based OTP systems are not connected to the client systems. This simplifies the tokens' deployment and makes them resistant to many kinds of malware attacks.

Impersonal authentication tokens can be used to complement hardware-based OTP systems in the sense that the resulting combined authentication system is SSL/TLS session-aware.[2] This is based on *U* employing the OTP as input for *f* (instead of $PIN_U$) in Equation 3. Hence, the UAC is computed as

$$UAC = f(N_T, OTP).$$

Everything else, including the construction of $N_T$, remains unchanged. This approach has the disadvantage that the user must have two tokens—the original OTP token and the impersonal authentication token—or the tokens must be integrated into one. The first possibility will likely be unacceptable in practice, whereas the second possibility will take time to implement because integrated tokens have yet to appear on the market.

For the different classes of OTP systems that we have listed, the first class is definitively the most difficult to make SSL/TLS session-aware. If we only have a physical list of OTPs, we can employ a technique similar to the one to make C/R systems SSL/TLS session-aware. If an SSL/TLS session between the browser and server is established, the user selects the next OTP to be used and enters it into the browser. The browser, in turn, interprets this value as an index into the *Hash* value. The

browser then extracts a specific number of bits from the *Hash* value and displays them. The bit sequence represents the UAC, which the browser displays and the user enters in a web form (instead of the OTP).

Many SSL/TLS-based e-commerce applications employ traditional authentication mechanisms on the client side. It is not widely known, however, 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 severity of this threat, and among those that have, many have declared client certificate authentication 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. A generally acceptable PKI-based solution that provides the degrees of freedom needed and supports the variety of platforms that customers require to use services like Internet banking has yet to be implemented.

These problems motivated us to propose TLS-SA on impersonal authentication tokens.[2] User authentication should not only depend on the user's secret credentials, but also on state information related to the SSL/TLS session in which credentials are transferred to the server. The technical feasibility of TLS-SA has been demonstrated in a prototype implementation that employs C/R tokens. More recently, we started work on making smart cards that conform to the SSL/TLS session-aware and can conform to the Eurocard, MasterCard, and Visa (EMV) Chip Authentication Program (CAP).[14] EMV-CAP debit cards will likely be widely deployed, at least in Europe, and will also be used for Internet banking.

Overall, TLS-SA offers a promising approach to solving the MITM problem. It leverages the currently used TLS/SSL implementations without alterations as well as the legacy authentication mechanisms and systems that the general public has become accustomed to using. It does not require a full-fledged PKI, as is mandatory when performing authentication using client certificates. Also, it does not inherit the privacy concerns that might arise if client certificates are used universally. Moreover, it can be deployed and operated at a fraction of the costs associated with a PKI-based solution. We thus believe that TLS-SA provides a lightweight privacy-enhancing alternative to deploying and using client certificates.

## References


1. T. Dierks and E. Rescorla, "The TLS Protocol Version 1.1," RFC 4346, Apr. 2006.
2. R. Oppliger, R. Hauser, and D. Basin, "SSL/TLS Session-Aware User Authentication—Or How to Effectively Thwart the Man-in-the-Middle," *Computer Comm.*, Aug. 2006, pp. 2238-2246.
3. A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," *Proc. CRYPTO 86*, LNCS 263, Springer, 1987, pp. 186-194.
4. P. Eronen and H. Tschofenig, eds., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)," Standards Track RFC 4279, Dec. 2005.
5. M. Badra and I. Hajjeh, "Key-Exchange Authentication Using Shared Secrets," *Computer*, Mar. 2006, pp. 58-66.
6. M. Steiner et al., "Secure Password-Based Cipher Suite for TLS," *ACM Trans. Information and System Security*, May 2001, pp. 134-157.
7. R.L. Rivest and A. Shamir, "How to Expose an Eavesdropper," *Comm. ACM*, vol. 27, no. 4, 1984, pp. 393-395.
8. S.M. Bellovin and M. Merritt, "An Attack on the Interlock Protocol When Used for Authentication," *IEEE Trans. Information Theory*, Jan. 1994, pp. 273-275.
9. N. Asokan, V. Niemi, and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols," *Proc. Int'l Workshop Security Protocols*, Springer-Verlag, 2003, pp. 15-24.
10. K.A. Murphy, "Gone Phishing," *Vantage*, vol. 2, no. 1, 2004, p. 17.
11. B. Parno, C. Kuo, and A. Perrig, "Phoolproof Phishing Prevention," *Proc. Financial Cryptography and Data Security*, Springer-Verlag, 2006, pp. 1-19.
12. A. Alkassar, C. Stüble, and A-R. Sadeghi, "Secure Object Identification—or: Solving the Chess Grandmaster Problem," *Proc. 2003 Workshop New Security Paradigms*, ACM Press, 2003, pp. 77-85.
13. R. Oppliger et al., "A Proof of Concept Implementation of SSL/TLS Session-Aware User Authentication," *Proc. Kommunikation Verteilten Systemen* (KiVS 2007), Springer-Verlag, 2007, pp. 225-236.
14. R. Oppliger and R. Hauser, "Protecting TLS-SA Implementations for the Challenge-Response Feature of EMV-CAP Against Challenge Collision Attacks," *Security and Communication Networks*, to appear.



***Rolf Oppliger*** *is the founder and owner of eSECURITY Technologies. His research interests include information security, cryptography, and cryptographic applications. Oppliger received a Habilitation in computer science from the University of Zurich. Contact him at rolf.oppliger@esecurity.ch.*

***Ralf Hauser*** *is a founder and the CTO of PrivaSphere AG. His research interests include messaging security, trust management, and voice security. Hauser received a PhD in computer science from the University of Zurich. Contact him at hauser@acm.org.*

***David Basin*** *is a professor of computer science at ETH Zurich. His research interests include information security, software engineering, and formal methods. Basin received a Habilitation in computer science from the University of Saarbrucken. Contact him at basin@inf.ethz.ch.*