A Proof of Concept Implementation of SSL/TLS Session-Aware User Authentication (TLS-SA)

Rolf Oppliger[1], Ralf Hauser[2], David Basin[3], Aldo Rodenhaeuser[4] und Bruno Kaiser[4]

[1] eSECURITY Technologies, Beethovenstrasse 10, CH-3073 Gümligen
[2] PrivaSphere AG, Jupiterstrasse 49, CH-8032 Zürich
[3] Department of Computer Science, ETH Zurich, Haldeneggsteig 4, CH-8092 Zürich
[4] AdNovum Informatik AG, Röntgenstrasse 22, CH-8005 Zürich

Abstract Most SSL/TLS-based e-commerce applications employ conventional mechanisms for user authentication. These mechanisms—if decoupled from SSL/TLS session establishment—are vulnerable to man-in-the-middle (MITM) attacks. In this paper, we elaborate on the feasibility of MITM attacks, survey countermeasures, introduce the notion of SSL/TLS session-aware user authentication (TLS-SA), and present a proof of concept implementation of TLS-SA. We think that TLS-SA fills a gap between the use of public key certificates on the client side and currently deployed user authentication mechanisms. Most importantly, it allows for the continued use of legacy two-factor authentication devices while still providing high levels of protection against MITM attacks.

1 Introduction

Most electronic commerce (e-commerce) applications in use today employ the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol [1] to authenticate the server to the client and to cryptographically protect the communication channel between them. When setting up a secure channel, the main point where SSL/TLS-based applications actually differ concerns user authentication. Conventional options available include passwords, personal identification numbers (PINs), transaction authorization numbers (TANs), scratch lists, as well as more sophisticated authentication systems, such as one-time password (OTP) and challenge-response (C/R) systems. In contrast, there are only a few applications that employ client-side public key certificates for user authentication as part of the SSL/TLS session establishment. In fact, the deployment of public key certificates has turned out to be slow—certainly slower than it was originally anticipated [2]. This is particularly true for client-side authentication.

In spite of the fact that many researchers have analyzed the security of the SSL/TLS protocol (e.g., [3, 4]), relatively few shortcomings and vulnerabilities have been identified (e.g., [5–7]. Most of them are theoretically interesting but not practically relevant. More problematic is the possibility that the protocol may be used in some inappropriate way or that it may be used in an environment that makes its security properties meaningless because the actual threats are

different than the ones assumed in the design phase (e.g., [8]). This paper further illustrates this point.

The vast majority of SSL/TLS-based e-commerce applications employ conventional user authentication mechanisms on the client side, and hence are vulnerable to phishing, Web spoofing, and—most importantly—man-in-the-middle (MITM) attacks [9].[1] These attacks are particularly powerful if visual spoofing is employed [10]. If a MITM can place himself between the user and the server, then he can act as a relay and authenticate himself to the server on behalf of the user. 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 often neglected when considering the security of SSL/TLS-based e-commerce applications, such as Internet banking or remote Internet voting.

In this paper, we elaborate on the feasibility of MITM attacks in Section 2, survey countermeasures and related work in Section 3, introduce the notion of SSL/TLS session-aware user authentication (TLS-SA) in Section 4, overview a proof of concept implementation of TLS-SA in Section 5, and discuss some weaknesses and limitations thereof in Section 6. Finally, we finish the paper with conclusions and an outlook in Section 7. In summary, we think that TLS-SA fills a gap between the use of public key certificates on the client side and currently deployed user authentication mechanisms. Most importantly, and in contrast to related work, it allows for the continued use of legacy two-factor authentication devices while still providing high levels of protection against MITM attacks.

2 MITM Attacks

According to RFC 2828, a *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." Consequently, the major characteristics of MITM attacks are that they represent active attacks, and that they target the associations between the communicating entities (rather than the entities or the communication channels). Note that in the literature, a MITM that carries out an active attack in real-time is sometimes also called *adaptive*. We will not use this term and MITM attacks will be adaptive by default.

In an SSL/TLS setting, which is standard in Internet banking and other Web-based e-commerce applications, the best way to think about a MITM attack is to consider an adversary that represents an SSL/TLS proxy server (or relay) between the client and the server. Neither the client nor the server are aware of the MITM. Cryptography makes no difference here as the MITM is in the loop and can decrypt and re-encrypt on the fly all messages that are sent back and

[1] According to http://www.theregister.co.uk/2005/10/12/outlaw_phishing, a MITM attack was launched against a Swedish Internet bank in November 2005. More recently, a MITM attack was launched against the Internet banking users of Citibank (as reported on July 10, 2006, by Brian Krebs in his Washington Post blog).

forth. Also, a MITM need not operate alone. In fact, there may be many entities involved in a MITM attack. One entity may be located between the client and the server, whereas some other entities may be located elsewhere. In this case, the corresponding attacks are called *Mafia* or *terrorist fraud* [11], and the underlying problem is referred to as the *chess grandmaster problem*. In this paper, we do not care whether the adversary is acting as a single-entity or multiple-entity MITM. Instead, we use the term MITM attack to refer to all types of attacks in which at least one entity is logically located between the client and the server.

In our SSL/TLS setting, there are many possibilities to implement a MITM attack. The user may be directed to the MITM using standard phishing techniques. Other possibilities include Address Resolution Protocol (ARP) cache poisoning, or Domain Name System (DNS) spoofing (including, for example, pharming). Anyway, MITM attacks may 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, for example, the user employs an OTP system, then the MITM can grab the OTP (which is typically valid for at least a few 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 [12], then the MITM would still be able to forward the messages and spoof the user. 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.[2]

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 reasons for this failure:

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

The first reason leads to a situation in which the user talks to the MITM, thereby revealing his credentials to the MITM. The second reason means that the credentials can be reused by the MITM to spoof the user to the origin server. We conclude that any effective countermeasure against MITM attacks in an SSL/TLS setting must address these problems by either enforcing proper server authentication or combining user authentication with SSL/TLS session establishment. The first possibility requires hard-coded server certificates or dedicated client software, whereas the second possibility requires modifications to the SSL/TLS or the authentication protocols in use. For the purpose of this paper, we confine ourselves to the second possibility and focus on modifying the authentication protocol accordingly.

[2] Note, however, that there is a construction [13] that can be used to make a zero-knowledge authentication protocol resistant against MITM attacks.

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: it is often stated within the security industry that strong (possibly two-factor) user authentication is needed to thwart MITM attacks.[3] This statement is flawed. 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 usually 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, given the sophistication of typical users. Note that a MITM can employ many tricks to give the user the impression of being connected to an origin server, for example using visual spoofing. In the most extreme case, one may think of a MITM that is able to control the graphical user interface of the browser. Also, we have seen phishing sites that employ valid certificates.[4] In such a setting, most users are out tricked and are unable to recognize that they are subject to a MITM attack.

Along the same line of argumentation, it is also important to note that the SSL/TLS protocol has been designed to natively protect users 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 have personal public key certificates. This requirement could be relaxed, if one extended the SSL/TLS protocol with some alternative client authentication methods (in addition to public key certificates). For example, there are efforts within the IETF to specify ciphersuites for the TLS protocol that support authentication based on pre-shared secret keys (e.g., [14, 15]) or OTP systems (e.g., [16]). Furthermore, the adoption of password-based key exchange protocols is proposed in [17], and the use of the Secure Remote Password (SRP) is specified in [18]. We think that these efforts are important, but there is a long way to go until the corresponding TLS extensions will be available, implemented, and widely 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 conventional authentication mechanisms.

In addition to the SSL/TLS protocol (and extensions thereof), there are several cryptographic techniques and protocols to protect users against MITM attacks:

[3] http://www.antiphishing.org/sponsors_technical_papers/PHISH_WP_0904

[4] 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 Mountain America Credit Union at www.mtnamerica.org.

- Rivest and Shamir proposed the Interlock protocol [19] that was later shown to be vulnerable when used for authentication [20].
- Jakobsson and Myers proposed a technique called delayed password disclosure (DPD) that can be used to complement a password-based authentication and key exchange protocol to protect against a special form of MITM attack—called the doppelganger window attack [21]. DPD requires a password-based authentication and key exchange protocol and is not qualified to protect against the kinds of powerful MITM attacks we have in mind.
- Kaliski and Nyström proposed a password protection module (PPM) to protect against MITM attacks [22]. The PPM is a trusted piece of software that utilizes password hashing to generate passcodes that are unique for a specific user and application. Again, PPMs do not provide protection against the kind of MITM attack we have in mind. Furthermore, PPMs appear difficult to deploy and manage.
- Asokan et al. proposed mechanisms to protect tunneled authentication protocols against MITM attacks [23]. These mechanisms are conceptually related to our approach.
- More recently, Parno et al. proposed the use of a trusted device (e.g., a Bluetooth-enabled smartphone) to perform mutual authentication and to thwart MITM attacks [24].

We believe that all of these techniques and protocols either do not adequately protect against MITM attacks or have severe disadvantages when it comes to a large-scale deployment.

Instead of cryptographic techniques and protocols, some researchers have suggested employing multiple communication channels and channel hopping to thwart MITM attacks (e.g., [25]). This approach has its own disadvantages and is impractical for almost all Internet-based applications in use today.

In practice, there is a steadily increasing number of applications—especially in Europe—that authenticate users by sending out SMS messages that contain TANs and require that users enter these TANs when they login. Sending out SMS messages is an example of using two communication channels or two-factor authentication (the mobile phone representing the second factor). While it has been argued that this mechanism protects against MITM attacks, unfortunately, this is not the case. If a 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.[5] For each transaction submitted by the user, a summary must be returned to the user together with a TAN in an SMS message. To confirm the transaction, the user must 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, and they are not even completely secure (a MITM can still attack the parts of a transaction

[5] http://www.cryptomathic.com/pdf/The Future of Phishing.pdf

that are not part of the summary). We have therefore investigated alternative mechanisms to protect users and their secure sessions against MITM attacks.

4 SSL/TLS Session-Aware User Authentication

Recall from Section 2 that an 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 (i.e., enforce proper server authentication), we think that this asks too much of the user. Hence, we confine ourselves to 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 approach. The basic idea of 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 behind this idea is that the server should have the possibility to determine whether the SSL/TLS session in which it receives the credentials is the same as the one the user employed when he sent out the credentials in the first place.

- 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 possible or even likely that a MITM is located between the client 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). A 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.

There are many possibilities to implement TLS-SA, and one can distinguish between hardware-based and software-based implementations.

- In the first case, we are talking about hardware tokens (i.e., hard-tokens). Such a token may be physically connected to the client system or not.[6]
- In the second case, we are talking about software tokens (i.e., soft-tokens).

In either case, an authentication token can be personal or impersonal, and it can be consistent with a cryptographic token interface standard, such as PKCS #11 or Microsoft's CAPI.

[6] We say that the token is physically connected to a system, or just connected, if there is a direct communication path in place between the token and the system. This includes, for example, galvanic connections, as well as connections based on wireless technologies, such as Bluetooth or infrared.

The basic approach to implement TLS-SA was introduced in [26]. It employs impersonal authentication tokens that users can plug into their client systems to support TLS-SA. If such a token is connected, then it is straightforward to provide support for TLS-SA (this is equally true for OTP and C/R systems). In this paper, however, we elaborate on the setting in which the token is not connected to the client system. In this case, the situation is more involved, because the token cannot directly access the hash value of the SSL/TLS session (that is stored in the browser's SSL/TLS cache). Note that there is nothing secret about the hash value (it can, for example, also be retrieved from the network traffic), but it still needs to be accessed in some well-defined way. An obvious possibility is to modify the browser so that it can display the hash value if needed (e.g., in the browser's status bar). Note, however, that the hash value is 36 bytes long, which is far too long to be used entirely (if the users must enter it manually). Consequently, one must reduce the length of the hash value to only a few digits to properly implement TLS-SA in such a setting.

5 Proof of concept Implementation

More recently, we have built a proof of concept implementation of TLS-SA. The aim is to demonstrate the technical feasibility of TLS-SA and to provide a testbed for analysis and further improvement. On the client side, the implementation employs C/R tokens from Vasco and a specially crafted plugin for Microsoft Internet Explorer (implemented as a CSP). On the server side, the implementation employs a modified version of AdNovum's Nevis Web portal. More specifically, it employs two Nevis components:

- A *secure reverse proxy* that is based on OpenSSL and Apache 2.0. OpenSSL is modified in the sense that it is possible to access and retrieve $Hash$ from the SSL/TLS session cache.
- A Java- and CORBA-based *authentication service* that is integrated with a Vasco backend component (i.e., the Vacman controller). The authentication service gets a UAC from the client system and verifies it (by recomputing it from the server-side SSL/TLS hash value).

The general idea to make a C/R-based authentication token (such as a Vasco token) SSL/TLS session-aware is to replace the challenges that are otherwise pseudorandomly generated by the server (and transmitted to the browser) with values that are derived from the SSL/TLS session state (e.g., hash value). These values can be generated independently by the server and the browser, and if the two are connected through the same SSL/TLS session, then the resulting values will be the same. Our TLS-SA proof of concept implementation is also based on this general idea. A message sequence diagram for the protocol that is implemented is illustrated in Figure 1. It consists of the following steps:

1. The user requests a URL by entering it into his browser.

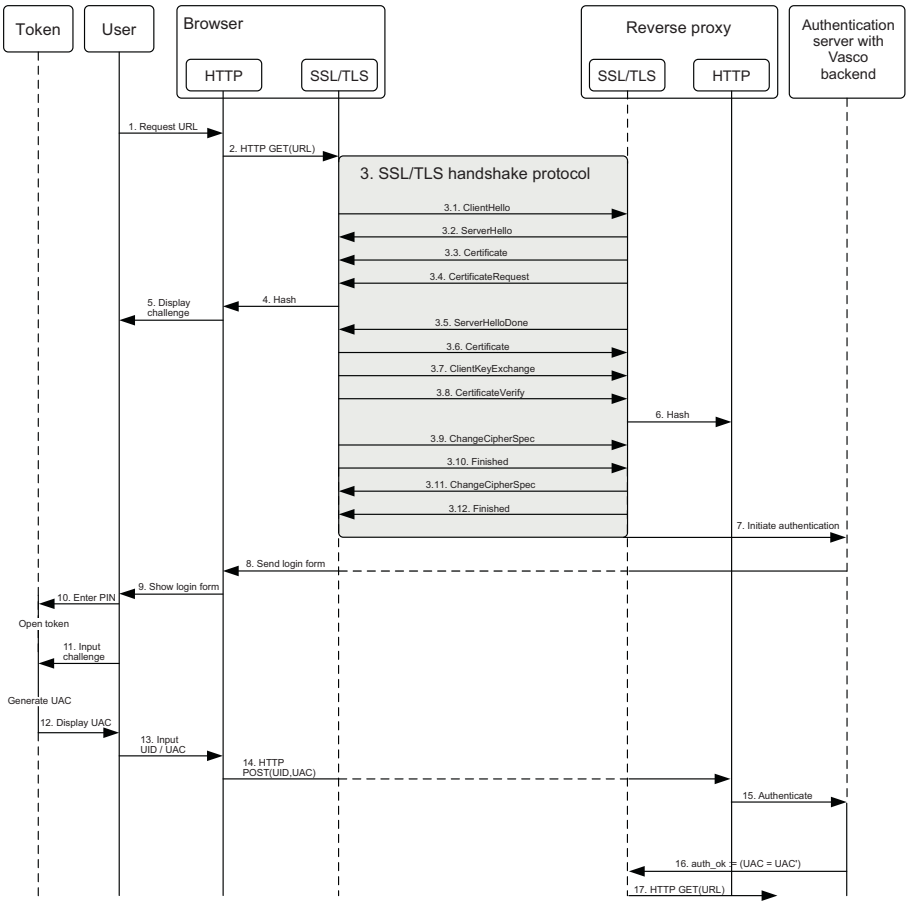

Figure 1 Message sequence diagram of our TLS-SA proof of concept implementation

2. The browser sends an HTTP GET request message for the URL to the Web server (or reverse proxy, respectively).
3. If the URL refers to a protected resource, then the Web server starts the SSL/TLS handshake protocol (according to the design principles of TLS-SA, we do not want the SSL/TLS handshake protocol to be changed). The server enforces certificate-based client authentication by sending out a CertificateRequest message to the browser in step 3.4. The browser, in turn, has a client certificate that is hardcoded (i.e., embedded) in the plugin. The certificate is sent to the server in step 3.6 as part of the Certificate message. In the CertificateVerify message of step 3.8, the browser proves knowledge of the private key. The bottom line is that the server can now be sure that the (anonymous) user employs the browser plugin and that the browser is bound to a particular SSL/TLS session (with a specific and bilaterally known

SSL/TLS hash value). For the protocol to work, it is not necessary that the private key remains secret, only its integrity is important or rather the entire plugin needs to be genuine.

4. The browser plugin grabs $Hash$ from the SSL/TLS session state.
5. The browser plugin displays the challenge $Challenge = encode(Hash)$, i.e., a user-friendly encoding and representation of $Hash$ to the user. Note that the encoding function *encode* is typically configurable by the user. Also note that the challenge is derived from the SSL/TLS session state, and that it is never sent from the server to the client. Because the Vasco token is only able to display decimal digits, the challenge must be encoded into this character set. This is achieved with a pseudorandom bit generator (PRBG) that is seeded with $Hash$. The output bit string is used to generate 4-bit sequences. Each 4-bit sequence is accepted if and only if it represents a decimal digit between 0 and 9 (otherwise, it is discarded). As soon as sufficiently many decimal digits are generated, the PRBG is stopped. In the current proof of concept implementation, the browser plugin can either display $Challenge$ in a popup window or write it as a message to standard output.
6. On the server side, a filter in the reverse proxy server grabs $Hash$ from the SSL/TLS session state and caches it together with the session ID. The server can trivially compute $Challenge$ from $Hash$ (using the *encode* function).
7. After having successfully executed the SSL/TLS handshake protocol, a secure channel between the browser and the reverse proxy is established. The reverse proxy checks whether the request is authenticated and forwards the unauthenticated request to the authentication service.
8. The authentication server sends a login form to the browser.
9. The browser displays the login form to the user.
10. The user enters his PIN into the token to unlock it.
11. The user enters the challenge as displayed in step 5 into the token.
12. The token calculates the response (i.e., the UAC) from the challenge and displays it to the user. The response comprises 7 decimal digits.
13. The user enters his user ID (UID) together with the UAC into the login form presented in step 9.
14. The browser sends the login form together with the user's credentials (i.e., UID and UAC) to the reverse proxy.
15. The reverse proxy forwards the user's credentials together with the challenge that was cached on the server side in step 6 to the authentication server.
16. The authentication server calculates the server-side response UAC' out of the server-side challenge and compares it with the client-side UAC sent by the client. Both responses must be identical in order to complete the authentication successfully. The authentication server sends back the result of the authentication (i.e., Boolean value `auth_ok`) to the reverse proxy.
17. If the authentication is successful (i.e., if `auth_ok` is `TRUE`), then the reverse proxy forwards the HTTP GET request message (as originally received in step 2) to the origin server.

Our proof of concept implementation demonstrates the basic idea of SSL/TLS session-aware user authentication and provides evidence of its feasibility. We point out and comment on several weaknesses and limitations next.

6 Weaknesses and Limitations

First, our proof of concept implementation has a weakness if the challenge is as short as described: the MITM can wait for a first SSL/TLS session to be established by the client. He can then set up a second SSL/TLS session to the origin server and modify the ServerHello message to be returned in the first session in a way that the deterministically compressed or truncated $Hash$ value matches the $Hash$ value of the second session. More specifically, the MITM looks for a second-preimage of $Hash$ for the compression or truncation function in use. In our case, this requires only 5,000 guesses on the average until a second preimage is found (provided that the challenge is 4 decimal digits long). So few guesses could be explored before the expiration time of the challenge. If the MITM has found such a second preimage, he can spoof the user by simply forwarding the user's response to the origin server. The solution we see (but have not implemented so far) to remedy this weakness is to have the browser pseudorandomly select a few bits from the 36 bytes long hash value, and to form a challenge that consists of these bits and indexes to their positions within the hash value. The C/R device then encrypts this challenge in a way that can be decrypted by the server, and sends the resulting response to the server. This turns the server into an online validation authority that can limit the number of false guesses. The disadvantage of this solution is that the response must at least be as long as the block size of the block cipher (e.g., 64 or 128 bits).

Second, our proof of concept implementation is incomplete: we only provide support for one C/R token, one browser, and one server. The most critical part is the browser because it is unrealistic to assume that users will change their browser only due to security concerns. Consequently, we would like to see browser enhancements to support TLS-SA (e.g., [27]). Most importantly, we would like to see browsers that are able to display $Hash$ or a user-friendly representation thereof (e.g., in the status bar). Enhancements like this can either be directly incorporated into a browser, or they can be implemented in the form of a browser plugin, as in our proof of concept implementation. It goes without saying that we prefer the first case. In the second case, the plugin must be professionally developed, integrated into a product, and deployed, all of which is nontrivial.

Third, TLS-SA and any implementation thereof does not protect against malware. If the client system hosts a Trojan horse or the browser is manipulated, then little can be done to protect the user.

7 Conclusions and Outlook

Many SSL/TLS-based e-commerce applications employ conventional user 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 personal certificates as the "strategic solution." To date, however, the often discussed obstacles to a full PKI rollout have prevented the successful, large-scale deployment of this solution.

Against this background, we think that TLS-SA provides a lightweight alternative to either a PKI[7] or the SSL/TLS extensions standardized by the IETF. Note that TLS-SA can work without modification of the SSL/TLS protocol. Also note that it is self-enforcing in the sense that users do not have to properly understand public key cryptography in order to be protected (*end user literacy*), and that the server can easily enforce the use of the protection mechanism (*enforcement and compliance*). This is in contrast to other protection mechanisms that rely on user education and proper browser configuration. Finally, note that TLS-SA has some privacy advantages, as it does not unambiguously identify the parties involved, also towards a illegitimately observing third party.

In this paper, we presented a proof of concept implementation of TLS-SA. This implementation is just one possibility to implement TLS-SA, and there are many other possibilities that may be explored. One particularly promising possibility is to make EMV-CAP[8] be SSL/TLS session-aware. In Europe, many financial institutions are providing their customers with EMV cards (to replace prior generations of payment cards). Hence, we expect most e-commerce users to possess one or more CAP-enabled EMV cards in the future. These cards are also intended for use in Internet banking. Under the conventional usage of SSL/TLS, however, they will be vulnerable to MITM attacks—with TLS-SA support they will not be.

References

1. Dierks T, Allen C: The TLS Protocol Version 1.0. RFC 2246, 1999.
2. Lopez J, Oppliger R, Pernul G: Why Have Public Key Infrastructures Failed so far? Internet Research, 15(5):544–556, 2005.
3. Mitchell J, Shmatikov V, Stern U: Finite-State Analysis of SSL 3.0. USENIX Security Symposium, 201–216, 1998.
4. Paulson LC: Inductive Analysis of the Internet Protocol TLS. ACM Trans. on Computer and System Security, 2(3):332–351, 1999.
5. Bleichenbacher D: Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. CRYPTO, 1–12, 1998.

[7] Of course the deployment of a PKI often has other objectives, e.g., the ability to provide nonrepudiation services.

[8] The EMV standard for smartcards was jointly developed and specified by Eurocard, MasterCard, and Visa. To use EMV cards for strong authentication in Internet-based e-commerce, MasterCard proposed the Chip Authentication Program (CAP) that was later adopted by Visa as Visa Passcode. It allows the user to generate OTPs or digital signatures.

6. Manger J: A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS#1 v2.0. CRYPTO, 230–238, 2001.
7. Vaudenay S: Security Flaws Induced by CBC Padding—Applications to SSL, IPSEC, WTLS ... EUROCRYPT, 534–545, 2002.
8. Anderson RJ: Why Cryptosystems Fail. Communications of the ACM, 37(11):32–40, 1994.
9. Burkholder P: SSL Man-in-the-Middle Attacks. SANS Reading Room, 2002.
10. Oppliger R, Gajek S: Effective Protection Against Phishing and Web Spoofing. CMS, 32–41, 2005.
11. Desmedt Y, Goutier C, Bengio S: Special uses and abuses of the Fiat-Shamir passport protocol. CRYPTO, 16–20, 1987.
12. Fiat A, Shamir A: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. CRYPTO, 186–194, 1986.
13. Cramer R, Damgård I: Fast and Secure Immunization Against Adaptive Man-in-the-Middle Impersonation. EUROCRYPT, 75–87, 1997.
14. Eronen P, Tschofenig H (Eds.): Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, 2005.
15. Badra M, Hajjeh I: Key-Exchange Authentication Using Shared Secrets. IEEE Computer, 39(3):58–66, 2006.
16. RSA Laboratories: OTP Methods for TLS. Draft 1, January 2006.
17. Steiner M., et al.: Secure Password-Based Cipher Suite for TLS. ACM Trans. Information and System Security, 4(2):134–157, 2001.
18. Taylor D, et al.: Using SRP for TLS Authentication. Work in progress, 2005.
19. Rivest RL, Shamir A: How to Expose an Eavesdropper. Communications of the ACM, 27(4):393–395, 1984.
20. Bellovin SM, Merritt M: An Attack on the Interlock Protocol When Used for Authentication. IEEE Trans. on Information Theory, 40(1), 1994.
21. Jakobsson M, Myers S: Stealth Attacks and Delayed Password Disclosure. 2005.
22. Kaliski B, Nyström M: Authentication: Risk vs. Readiness, Challenges & Solutions. BITS Protecting the Core Forum, October 6, 2004.
23. Asokan N, Niemi V, Nyberg K: Man-in-the-Middle in Tunneled Authentication Protocols. International Workshop on Security Protocols, 15–24, 2003.
24. Parno B, Kuo C, Perrig A: Phoolproof Phishing Prevention. Financial Cryptography, 2006.
25. Alkassar A, Stüble C, Sadeghi AR: Secure Object Identification—or: Solving The Chess Grandmaster Problem. Workshop on New Security Paradigms, 77–85, 2003.
26. Oppliger R, Hauser R, Basin D: SSL/TLS Session-Aware User Authentication—Or How to Effectively Thwart the Man-in-the-Middle. Computer Communications, 29(12):2238–2246, 2006.
27. Oppliger R, Hauser R, Basin D: Browser Enhancements to Support SSL/TLS Session-Aware User Authentication. W3C Workshop on Transparency and Usability of Web Authentication, 2006.