SSL/TLS Session-Aware User Authentication Using a GAA Bootstrapped Key

Chunhua Chen[1⋆], Chris J. Mitchell[2], and Shaohua Tang[3]

[1,3]School of Computer Science and Engineering
South China University of Technology
Guangzhou 510640, China
[1]chunhua.chen@mail.scut.edu.cn, [3]csshtang@scut.edu.cn
[2]Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
c.mitchell@rhul.ac.uk

Abstract. Most SSL/TLS-based electronic commerce (e-commerce) applications (including Internet banking) are vulnerable to man in the middle attacks. Such attacks arise since users are often unable to authenticate a server effectively, and because user authentication methods are typically decoupled from SSL/TLS session establishment. Cryptographically binding the two authentication procedures together, a process referred to here as SSL/TLS session-aware user authentication (TLS-SA), is a lightweight and effective countermeasure. In this paper we propose a means of implementing TLS-SA using a GAA bootstrapped key. The scheme employs a GAA-enabled user device with a display and an input capability (e.g. a 3G mobile phone) and a GAA-aware server. We describe a simple instantiation of the scheme which makes the password authentication mechanism SSL/TLS session-aware; in addition we describe two possible variants that give security-efficiency trade-offs. Analysis shows that the scheme is effective, secure and scalable. Moreover, the approach fits well to the multi-institution scenario.

Keywords: man in the middle, SSL/TLS session-aware user authentication, Generic Authentication Architecture

1 Introduction

Most current e-commerce applications (including Internet banking) employ the Secure Socket Layer (SSL) [12] or the Transport Layer Security (TLS) protocol[1] [9] to cryptographically protect the communication channel between the

⋆ The author is a PhD student at the South China University of Technology. This work was performed during a visit to the Information Security Group at Royal Holloway, University of London, sponsored by the Chinese Scholarship Council and the Natural Science Foundation of Guangdong Province, China (No. 9351064101000003).

[1] The minor differences between SSL and TLS are not relevant here, and we thus refer to them jointly as SSL/TLS throughout.

client and the server. Typically, when establishing the SSL/TLS session, the server authenticates itself to the client using a public key certificate. Although the client could also authenticate itself to the server using a public key certificate (an option in the SSL/TLS protocol), in practice this rarely takes place since very few clients have the necessary key pair and certificate [19]. Instead, SSL/TLS-based applications typically employ a separate user authentication protocol on top of SSL/TLS, e.g. using a password, personal identification number (PIN), or a more sophisticated mechanism such as a one-time password system.

The SSL/TLS protocol appears reasonably sound, and the security issues so far identified [20, 26] appear to be relatively minor. However, in practice, the SSL/TLS protocol does not provide a high level of security because it requires the user to verify with whom the client system is communicating, a task that is often poorly performed [8]. Man in the middle attacks arise precisely because of this shortcoming and the fact that the user authentication process is decoupled from SSL/TLS session establishment. Consequently, any effective countermeasure against these man in the middle attacks in an SSL/TLS setting must either enforce proper server authentication or combine the user authentication process with SSL/TLS session establishment. Oppliger et al. [23, 25] introduced the term SSL/TLS session-aware user authentication (TLS-SA) to describe the latter countermeasure, and proposed an approach of this type using a non-user-specific hardware token that shares a secret key with the server. However, this pre-shared key based approach has a number of disadvantages (see also section 3).

Universal Mobile Telecommunications System (UMTS) networks have been widely deployed, and there are a huge number of subscription holders across the world. The Generic Authentication Architecture (GAA) [3] exploits the UMTS authentication infrastructure to enable the provision of security services, including key establishment, to third party mobile and Internet applications. In essence, GAA makes use of the UMTS Authentication and Key Agreement (UMTS AKA) protocol [2] to bootstrap application-specific session keys between GAA-enabled devices and GAA-aware servers.

To avoid the disadvantages of the pre-shared key based approach, we propose a means of implementing TLS-SA using a GAA bootstrapped key. The scheme employs a GAA-enabled user device with a display and an input capability (e.g. a 3G mobile phone) and a GAA-aware server, and binds the user authentication process to the TLS session without modifying the operation of TLS. Analysis shows that the scheme is effective, secure, scalable and has a degree of flexibility enabling security-efficiency trade-offs. Moreover, the approach fits well to the multi-institution scenario. The rest of this paper is organised as follows. In section 2 we describe relevant background. In section 3 we survey related work with an emphasis on TLS-SA. In section 4 we propose the GAA-based approach, and also describe two possible variants that give security-efficiency trade-offs. In section 5 we present a security analysis, and in section 6 we draw conclusions.

2 Background

In this section we briefly describe the man in the middle attacks relevant to SSL/TLS-based applications, as well as relevant details of UMTS AKA and GAA.

2.1 Man in the Middle Attacks

A man in the middle attack targets associations between communicating entities. Typically, an adversary places itself between the client and the server and establishes separate associations with them. It then intercepts and selectively modifies communicated data to masquerade as the legitimate entities. Cryptographic protection does not in itself prevent such an attack, because the adversary engages in association establishment and possesses all the negotiated cryptographic keys.

We assume the following SSL/TLS setting in this paper:

1. the SSL/TLS protocol is only used to authenticate the server, and
2. user authentication is performed using username and password via an established SSL/TLS session.

When establishing the SSL/TLS tunnel, the user should verify the identity of the remote system with which the client system (e.g. a browser) is communicating. If not, then certain man in the middle attacks become possible.

Some classes of phishing attacks are examples of such man in the middle attacks, and attacks of this type have become widespread [1]. In these attacks, an adversary sets up a fake web site which imitates an existing legitimate site in order to mislead users and obtain their authentication credentials. Dhamija et al. [8] show that users often cannot distinguish a legitimate web site from a fake (including in the case where SSL/TLS server authentication is employed).

Against this background, and as discussed by Oppliger et al. [23], most currently deployed user authentication mechanisms fail to provide effective protection against man in the middle attacks, even when they run over the SSL/TLS protocol. There are two main reasons for this.

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

If both the above assumptions hold, an attacker can first establish a SSL/TLS session with the client, and fool the user into revealing his or her credentials. The attacker then establishes a separate SSL/TLS session with the server it has impersonated, and masquerades as the user by retransmitting the stolen credentials. Defeating this man in the middle attack requires either proper server authentication or a means of combining the user authentication process with SSL/TLS session establishment.

2.2 Generic Authentication Architecture

The information in this section is mainly derived from Holtmanns et al. [14].

The UMTS AKA protocol provides authentication and key establishment using a long-term secret subscriber key (K), shared by a user device (e.g. a 3G mobile phone) and a mobile network. After a successful UMTS AKA procedure, a pair of secret session keys is established which are shared by the device and the network. The established keys are CK, used for confidentiality protection, and IK, used for integrity protection. We note also that, in the UMTS AKA procedure, a random challenge ($RAND$) is sent by the mobile network to the user device.

The Generic Authentication Architecture (GAA) [3] has been standardised by the 3rd Generation Partnership Project (3GPP), and its North American counterpart, the 3rd Generation Partnership Project 2 (3GPP2). The 3GPP standard versions of GAA build on the widely established mobile authentication infrastructures (including the GSM and UMTS infrastructures). In this paper we focus on GAA as supported by the UMTS authentication infrastructure. GAA consists of two procedures: GAA bootstrapping and Use of bootstrapped keys.

GAA bootstrapping, also known as the Generic Bootstrapping Architecture (GBA), is the process by which UMTS AKA is used to set up a GAA master session key (MK) between a GAA-enabled device and a network, where MK is the concatenation of IK and CK. The network also sends a transaction identifier $B\text{-}TID$. $B\text{-}TID$ is generated from the $RAND$ value and the network domain name of the mobile network, and can be used to identify MK and its lifetime to the GAA-enabled device. Both the GAA-enabled device and the network cache MK, the lifetime of MK and $RAND$ for later use. The master session key MK is not bound to a particular application, and can only be used to derive application-specific session keys.

Use of bootstrapped keys is the procedure by which a GAA-enabled device employs the bootstrapped keys to secure its exchanges in an application protocol with a particular GAA-aware application server. Once the GAA-enabled device decides to engage in an application protocol with a particular GAA-aware server, it derives an application-specific session key (SK) from MK, as follows:

$$SK = \mathrm{KDF}(MK, \mathrm{GBAvariant}, RAND, \mathrm{IMPI}, NAF\text{-}Id)$$

where KDF is a key derivation function, GBA_variant indicates the bootstrapping variant (such as GBA_ME or GBA_U), the IP Multimedia Private Identifier (IMPI) is derived from the International Mobile Subscriber Identity (IMSI) [6] which is unique to each mobile phone, and $NAF\text{-}Id$[2] is an application-specific value consisting of the Fully Qualified Domain Name (FQDN) of an application server and the identifier of the underlying application protocol. The device starts the application protocol by sending a request containing $B\text{-}TID$. The server fetches the same SK, the lifetime of SK, and other relevant information

[2] In the GAA specifications [3], the functionality of a GAA-aware application server is referred to as the Network Application Function (NAF).

from the corresponding mobile network by forwarding the received *B-TID* and its own identifier *NAF-Id*. Note that *B-TID* contains the network domain name of the mobile network, so the application server knows where to send the request. Normally the network has to authenticate that the requesting server is the genuine owner of FQDN, which forms a part of *NAF-Id*. In GAA, it is assumed that a confidential and authenticated channel between the server and the network has been set up by some means. At this point, the device and the server share the same value of SK. It is important to note that SK is bound to a specific application protocol and a particular application server.

In summary, UMTS GAA uses UMTS AKA to bootstrap application-specific session keys between GAA-enabled devices and GAA-aware servers.

3 Related Work

The incorporation of password authenticated key exchange (PAKE) schemes into the TLS protocol has been proposed by Steiner et al. [27] and, subsequently, by Abdalla et al. [7]. More recently, the use of the Secure Remote Password (SRP) protocol within TLS has been specified in an Internet draft [28]. Despite the potential advantages of such an approach, migrating from legacy user authentication to a PAKE-based system is non-trivial for a variety of technical and business reasons [10].

An application of GAA based on Pre-Shared Key (PSK) TLS [11] has been described in 3GPP documents [4, 5]. GAA credentials are used to establish a TLS session by setting the Pre-Shared Key identity to be the *B-TID*, and the PSK to be the application-specific session key. Note the PSK TLS protocol is able to protect against man in the middle attacks.

Oppliger et al. [23] introduced SSL/TLS session-aware user authentication (TLS-SA), a lightweight and effective countermeasure [24] to man in the middle attacks. TLS-SA makes user authentication depend not only on the user's secret credentials, such as his or her password, but also on SSL/TLS session state information. As a result, the server can check whether or not the SSL/TLS session in which it receives the credentials matches the one employed by the user to send them. If the two sessions match, it is unlikely that a man in the middle is involved; however, if they differ, something abnormal must be taking place, e.g. a man in the middle attack is being performed.

TLS-SA is not a user authentication mechanism or system. Many different approaches can be used to make a given authentication mechanism SSL/TLS session-aware and hence resistant to man in the middle attacks. Oppliger et al. [23] proposed a pre-shared key based approach, and subsequently described a proof of concept implementation [25]. This scheme involves a hardware authentication token which shares a secret key with the server.

One disadvantage of this pre-shared key based approach is that every server needs to generate and securely distribute a key-bearing token to every user, which is likely to be a significant burden in practice. Another disadvantage is its poor scalability. A subsequent proposal [24] involves the use of a multi-institution

token which is equipped with a separate secret key for each of a number of servers; however such a scheme may be difficult to market.

4 TLS-SA Using a GAA Bootstrapped Key

We now propose a means of implementing TLS-SA using a GAA bootstrapped key. The scheme employs a GAA-enabled user device with a display and an input capability (e.g. a 3G mobile phone) and a GAA-aware server. During the user authentication process, an application-specific session key (SK) is bootstrapped between the device and the server using GAA. The device uses this GAA bootstrapped key to compute a user authentication code from a combination of the user's secret credentials and state information for the current SSL/TLS session. The user authentication code is submitted to the server to authenticate the user, instead of the secret credentials. It is important to observe that, unlike previously proposed TLS-SA schemes, the system we describe does not require an initialisation process. That is, the GAA-enabled mobile device is not user or server specific, and can be used in the protocol with no registration or configuration (except for the installation of the necessary application software).

The state information to be used in the computation of user authentication code must have the following properties: (1) it must be shared by the client and the server and be distinct for every SSL/TLS session, and (2) the state information established by a server operated by a man in the middle attacker must be different from the value established by the genuine server. In the scheme described below, this is achieved by using as state information a hash of all the messages exchanged during the underlying SSL/TLS Handshake, computed using a suitable cryptographic hash function.

In the remainder of this section we first describe a simple instantiation of the scheme. We then discuss two possible variants that give security-efficiency trade-offs.

4.1 The Basic Scheme

The following entities play a role in the scheme:

- A user U.
- A SSL/TLS-enabled and GAA-aware server S. We assume that the application supporting the scheme and executing in S can access certain elements of the SSL/TLS session information.
- A SSL/TLS-enabled client (e.g. a browser) C, used by U to access S. We assume that an application supporting the scheme has been installed in C (e.g. as a Java applet/browser plug-in). This application must have ability to access certain elements of the client's SSL/TLS session information and be aware of the FQDN of the underlying S and the identifier of the underlying application protocol.

- A GAA-enabled user device T (e.g. a 3G mobile phone) with a display and an input capability. We assume that an application supporting the scheme has been installed in T. This application must possess a means of communication with the scheme-specific application in C in order to get the necessary information, e.g. as provided by a USB cable or a Bluetooth link.
- A mobile network provider N that provides the GAA service, and that is trusted by users and servers.

These entities are equipped with various parameters and cryptographic keys. U is equipped with an identifier *username* and a *password* (*pw*), a secret shared with S. A long-term secret subscriber key (K) is shared by T (strictly its USIM) and its home mobile network as part of the subscription. Note that we also assume that S has the means to establish a secure authenticated channel (e.g. as provided by an SSL/TLS tunnel) with a mobile network as necessary to use GAA.

1. $C \leftrightarrow S$: establish an SSL/TLS session,
   : and generate H.
2. $T \leftrightarrow N$: *B-TID*, *MK*, *RAND*
   : and the lifetime of *MK* ($[T_{start}, T_{end}]$).
3. $C \rightarrow T$: H, the FQDN of S, and the identifier
   : of the application protocol.
4. T : derives a session key *SK*.
5. T : computes $uac = f(H, SK, pw, \ldots)$.
6. $U(C) \rightarrow S$: *B-TID*, *username*, and *auc*.
7. $S \leftrightarrow N$: *SK* and the lifetime of *SK* ($[T_{start}, T_{end}]$).
8. S : $T_{current} \in [T_{start}, T_{end}]$?
   : if so, S recomputes *auc* for authentication;
   : if not, S discards the request.

Figure 1: the GAA-based TLS-SA protocol

Figure 1 summarises the GAA-based TLS-SA protocol. We next give a more detailed description, referring to the step numbers shown in the figure.

When U wishes to access S, U directs its client C to S. C and S then establish an SSL/TLS session with server authentication using a public key certificate (step 1).[3] Once the session has been established, C and S compute (and cache)

$$H = h(Msgs)$$

where h refers to a suitable cryptographic hash function, such as SHA-1 or one of the SHA-2 family [21] and *Msgs* denotes all the messages exchanged within the SSL/TLS session establishment process.[4]

[3] Whether or not the user verifies that S is indeed the server it wishes to communicate with is not critical to the security of the scheme.

[4] Enabling the application to gain access to these messages may require minor modifications to the SSL/TLS implementation. However, it requires no change to the SSL/TLS Handshake protocol itself.

T next checks whether it has a pre-established and valid master session key MK. If not, T triggers a GAA bootstrapping procedure with its home network. After successful execution of this process, the values *B-TID*, MK, the lifetime of MK, and *RAND* are shared and cached by T and its home network (step 2).

U employs T to communicate with the scheme-specific application in C to get H, the FQDN of S and the identifier of the application protocol (step 3). T then constructs *NAF-Id* and derives a session key SK, as described in section 2.2 (step 4). Note that SK is not specific to U, and cannot be used to authenticate U to S.

After derivation of SK, T uses U's password pw (which must be input by U at some point) to compute a user authentication code uac as a function f of SK, H, pw, and other relevant parameters (step 5), i.e.

$$uac = f(SK||H||pw||\ldots),$$

where here and throughout $||$ is used to denote concatenation. The function f can be implemented in many ways. One possibility, which complies with clause 5.1.2 of ISO/IEC 9798-4 [16], is to instantiate f using HMAC [18] based on a suitable cryptographic hash function, where the various inputs to f are simply concatenated prior to applying HMAC. In this case, uac is computed as:

$$uac = \mathrm{HMAC}_{SK}(H||pw||\text{“Client”}).$$

As discussed above, H is an SSL/TLS session-specific value, and it plays the role of the nonce in the ISO/IEC 9798-4 protocol. The fixed string “Client” plays the role of the entity identifier.

The server authenticates the user U by asking him or her to submit the values *B-TID*, *username*, and uac using the SSL/TLS-protected channel (step 6). Note that the user is not required to enter these values into the client, since they can be transferred electronically from T to C. To verify the received uac, S fetches the same SK, the lifetime of SK, and other relevant information from T's home network using the GAA bootstrapped key usage procedure (step 7). SK's lifetime can be set to be the same as that of MK. Before recomputing uac, S must check whether or not SK is valid. This is achieved by checking whether or not the current system time of S is within SK's lifetime.[5] If not, SK is invalid and U will be rejected; otherwise, S can now use the received SK to recompute uac for verification.[6] If the recomputed uac and the uac submitted by U match, U will be granted access (step 8).

Note that the SSL/TLS implementations in C and S need to provide access to session information to the application layer [13, 17]. We propose that the SSL/TLS implementations compute H upon the completion of SSL/TLS Handshake session establishment, and cache it as part of the connection state for the SSL/TLS Record layer. However, how this is achieved is application and SSL/TLS implementation specific, and hence we do not discuss this further here.

[5] Note that we assume that the system time of the network and the server are synchronised with each other.

[6] S must retrieve H at some point.

4.2 Variants

In GAA, *SK* is typically used as a session key to secure application data sent between client and server. Using *SK* instead of *MK* reduces the risk of disclosing the master key *MK*, e.g. as a result of cryptanalysis. However, in our case *SK* is only used in the computation of the user authentication code, i.e. only a small amount of data is involved. Other application data is protected by transmission through an SSL/TLS-protected channel.

Examining the computation of *SK* (as above), it follows that, during the lifetime of the key *MK*, the value of *SK* only depends on *NAF-Id*. In practice, *U* could repeatedly access a particular server using the same application protocol.[7] In such a case, *U* would use the same *NAF-Id* in the derivation of *SK* in multiple user authentication sessions. In fact, the GAA bootstrapping procedure can be avoided after the first use by setting the lifetime of *MK* to be sufficiently long. In this case, *T* uses the same *MK* in the derivation of *SK* in all subsequent authentication sessions, and hence a user can employ the same *SK* for all authentication sessions with a particular server which involve a specific application protocol. As a result, it is reasonable to propose that *T* and *S* both cache *SK* and use it as a long-term shared key to avoid frequent use of GAA and derivations of *SK*. We next describe two variants to achieve this.

In both variants, *T* and its home network first carry out a GAA bootstrapping procedure to establish a shared master key *MK* and other information. The lifetime of *MK* must be set to be sufficiently long. This process is performed before any user authentication processes.

A Straightforward Variant The user device *T* can identify *SK* using *NAF-Id*, which is constructed from the FQDN of *S* and the identifier of the application protocol. When computing *uac*, *T* first tries to retrieve *SK* from its cache using the constructed *NAF-Id* as index. If *SK* is not present, then *T* derives an *SK* from *MK* and the constructed *NAF-Id*, stores the pair (*NAF-Id*, *SK*), and then uses the derived *SK* to compute *uac*. If *SK* is in the cache, *T* uses it directly in the computation of *uac*.

The server *S* needs to use both *B-TID* and *NAF-Id* to identify *SK*. *U* submits *username*, *B-TID*, and *uac* in the authentication process. Upon receiving an authentication request from *C*, *S* tries to retrieve *SK* from its cache using the received *B-TID* and its own *NAF-Id* as index. If *SK* is not present, *S* requests it from *T*'s home network, stores the received triple (*B-TID*, *NAF-Id*, *SK*), and then verifies the *uac* to authenticate the user. If *SK* is present, *S* uses it directly in the verification process.

A Separate Registration Procedure Variant In the basic scheme and the first variant described above, *U* has to submit *username*, *B-TID*, and *uac* in order to be authenticated by *S*. *B-TID* is submitted so that *S* can identify

[7] For example, the user might repeatedly interact with a browser to access his or her bank account via HTTPS.

SK. In a standard password-based user authentication process, only *username* and *pw* need to be submitted. Thus submitting *B-TID* potentially significantly increases the traffic load (since *B-TID* is much longer than a typical password), and increases the complexity of implementation in settings in which *B-TID* must be input by U. In the variant we now describe, S is required to provide a service which allows U to register *SK*, enabling S to identify *SK* from just the *username*. As a result, U will only need to submit *username* and *uac* in the authentication process. The registration of *SK* can be done using a registration procedure of the type described below.

Before any user authentication process, T derives *SK* to be used by U to access a particular server S through a specified application protocol. T stores the pair (*username*, *SK*). If $M = H||pw$, then, in order to register *SK*, U submits *username*, *B-TID*, and $E_{SK}(M)$ via a previously established SSL/TLS-protected channel. Here E is an authenticated encryption technique, e.g. one of those standardised in ISO/IEC 19772 [15]. Upon receiving the request, S fetches the same *SK*, the lifetime of *SK* and other relevant information from T's home network using the GAA bootstrapped key usage procedure. S can now use the received *SK* to decrypt and verify the encrypted version of M to recover H and *pw*. S then checks whether the received H matches the current SSL/TLS session. Finally, S verifies whether *username* identifies a valid user and *pw* is the correct password for this user. If all the verifications succeed, then S registers the binding between *username* and *SK*.

After a successful registration procedure, T and S share the same *SK* which can be identified by *username*. Thus, in the user authentication process, U only needs to submit *username* and *uac*. However, it is important to note that the binding between U and *SK* remains weak, and is only useful for the purpose of identifying *SK*. That is, successful user authentication will require knowledge of both *SK* and *pw*.

5 Analysis

The GAA-based approach avoids the disadvantages of the pre-shared key approach discussed in section 3. To implement the GAA-based approach, the user only needs a GAA-enabled device with a display and an input capability. This can be implemented using a 3G mobile phone with a valid subscription, and there are a very large number of subscription holders across the world. The approach thus has good scalability. Moreover, since a GAA bootstrapped session key is used in the computation of the user authentication code, there is no need to generate and securely distribute a key-bearing token to every user. The approach also fits well to the multi-institution scenario. The system enables server-specific session keys to be generated using a single GAA-enabled device, where each such key can be used to help authenticate a user to the appropriate GAA-aware server. The GAA-enabled device thus acts as a non-institution-specific authentication token.

However, in deciding whether to use this GAA-based TLS-SA system, the server S and its users must trust the mobile network provider not to compromise its long-term password (see also section 5.1). Such a trust relationship could be supported by a contractual agreement between application service providers and mobile operators.

A limitation of the scheme is that use of the system will incur the cost of using the GAA service. The two variants described in section 4.2 are more cost-effective in this respect than the basic scheme.

We next give an informal security analysis. We then show that the GAA-based approach has a degree of flexibility, enabling the implementation of security-efficiency trade-offs.

5.1 Informal Security Analysis

We consider a threat model in which an attacker $\mathcal{A}$ is able to observe and make arbitrary modifications to messages exchanged between C and S, including replaying, blocking and inserting completely spurious messages. This allows a trivial denial of service attack which cannot be prevented. $\mathcal{A}$ is also assumed to be a legitimate user of the UMTS GAA service. However, $\mathcal{A}$ is not allows to compromise the implementations of T, C or S (e.g. using malware); such attacks on system integrity are not addressed by the schemes we propose.

The security of the schemes relies on the security of the underlying UMTS GAA and SSL/TLS protocol. In turn, the security of UMTS GAA is built on the assumption that learning the subscriber key and/or MK by attacking UMTS AKA is not possible [14].

We next provide a brief informal analysis of how the schemes meet the intended security goals.

1. *Resistance to user authentication code replay.*
   The GAA-based scheme, like the pre-shared key approach, involves authenticating a user via a user authentication code (*uac*). The user authentication code is cryptographically bound to the current SSL/TLS Handshake session state information. The state information (H) is a cryptographic hash of all the messages exchanged during SSL/TLS session establishment.
   Suppose $\mathcal{A}$ launches a man-in-the-middle attack by establishing two separate SSL/TLS sessions: one with S (masquerading as C) and one with C (masquerading as S). The value of the *uac* provided by C to $\mathcal{A}$ will be a function of the messages exchanged by C and $\mathcal{A}$ during SSL/TLS session establishment; similarly the *uac* expected by S will be a function of the messages exchanged by $\mathcal{A}$ and S during SSL/TLS session establishment. Even if they are otherwise identical, the first set of messages will include $\mathcal{A}$'s SSL/TLS server certificate, and the second set of messages will instead contains S's certificate. As a result the *uac* provided by C will be different to that expected by S, and hence the attack will fail.
2. *Resistance to compromise of a user password (pw).*

$\mathcal{A}$ could set up an SSL/TLS session with C (impersonating a legitimate S) and request a *uac*. The *uac* is computed using a keyed one way hash function which takes as input the GAA bootstrapped session key; it is thus infeasible to retrieve pw from a valid *uac* without knowing this session key. That is, $\mathcal{A}$ cannot succeed in a off-line dictionary attack against pw without knowing the corresponding SK or MK (using MK, $\mathcal{A}$ can derive SK). Similarly, in the registration procedure, $\mathcal{A}$ cannot retrieve pw from the encrypted string sent to S without knowing the corresponding SK or MK. We assume that the underlying UMTS GAA is secure [22], that is, it is impossible for $\mathcal{A}$ to learn SK (intended for a legitimate S) or MK established between T and its mobile network by attacking UMTS GAA.
Alternatively, $\mathcal{A}$ could attack the KDF algorithm used for SK derivation. In such an attack, $\mathcal{A}$ chooses and registers a value for *NAF-Id* (*NAF-Id*$_{\mathcal{A}}$, say) which has the property that

$$\text{KDF}(MK, \text{GBAvariant}, RAND, \text{IMPI}, \textit{NAF-Id}_{\mathcal{A}}) = \text{KDF}(MK, \text{GBAvariant}, RAND, \text{IMPI}, \textit{NAF-Id}_S)$$

for any master key MK, where *NAF-Id*$_S$ is the *NAF-Id* for a legitimate server S. $\mathcal{A}$ then requests an SK from the mobile network by sending *B-TID* and *NAF-Id*$_{\mathcal{A}}$. The mobile network derives an SK from *NAF-Id*$_{\mathcal{A}}$, which is equal to the SK derived from *NAF-Id*$_S$. As a result, the adversary learns the value of SK for U and S. However, the KDF algorithm used in GAA is based on HMAC-SHA-256, which is believed to be a secure MAC function [14], and hence such a collision attack is believed to be infeasible. The choice of a cryptographically strong KDF also means that even if $\mathcal{A}$ has discovered a number of SK values, they cannot be used to discover other keys derived from the same master key MK.

3. *Resistance to registration message replay (second variant only).*
   Note that S can detect such an attack by checking whether the H value matches the current SSL/TLS session. If not, S simply discards the request.

It is very important to note that the mobile network operator possesses the GAA bootstrapped session keys and must be trusted since, if it obtains the authenticator[8], it could perform a dictionary attack to find the user's long-term password. In practice, users already trust mobile operators not to intercept their phone calls. This is a high level of trust since operators could, for example, intercept and misuse a wide range of user secrets (e.g. credit card details).

5.2 Security-efficiency Trade-offs

In the pre-shared key approach outlined in section 3, a secret key is used in all authentication sessions. This is arguably more efficient, since there is no need

[8] A malicious mobile network provider could potentially set up a phishing website to try to persuade a user to submit an authenticator. However, such a scenario seems rather far-fetched, particularly given that a user can choose which network operator to use.

for session key establishment (e.g. a GAA procedure) in the user authentication process. However, in a high security scenario it may be necessary to use a new key for each authentication session. Secret key re-configuration is highly non-trivial in the pre-shared key approach, since tokens have to be distributed to users. As we show immediately below, the GAA-based approach has the flexibility to enable security-efficiency tradeoffs.

For a high-security scenario in which a new session key SK is needed for each authentication session, the basic GAA-based scheme can be used. By setting the lifetime of MK to be sufficiently short, a new master MK will be established between T and its home network in each authentication session. As a result, a new session key SK will be derived for the computation of each user authentication code. Of course, this will introduce a significant network traffic and computational overhead, including the establishment of MK using the GAA bootstrapping procedure, the derivation of the session key SK in T, and the fetching of SK from the mobile network by S. Such an approach therefore has relatively high security and low efficiency. In a scenario where security is not quite such a high priority, one of the variant schemes can be used in which the key SK is cached and used as a long-term key for multiple user authentication sessions. As a result, additional GAA procedures and calculations are avoided, and higher efficiency can be achieved.

6 Conclusions

SSL/TLS session-aware user authentication is a lightweight and effective countermeasure to man in the middle attacks. We propose a means of implementing SSL/TLS session-aware user authentication using a GAA bootstrapped key. The scheme employs a GAA-enabled user device with a display and an input capability (e.g. a 3G mobile phone) and a GAA-aware server. Importantly, the user device does not need to be registered with the server, and no server-specific details are stored in the device; that is, the user device is not specific to either the user or the server. Analysis shows that the scheme is effective, secure, scalable and has a degree of flexibility enabling security-efficiency trade-offs. Moreover, the scheme fits well to the multi-institution scenario.

References

1. Anti-phishing working group phishing archive. http://anti-phishing.org/phishing_archive.htm.
2. 3rd Generation Partnership Project (3GPP). *3G Security: Access Secure for IP-based Services, Version 9.3.0*, 2009.
3. 3rd Generation Partnership Project (3GPP). *3rd Generation Partnership Project, Technical Specification Group Services and Systems Aspects, Generic Authentication Architecture (GAA), Generic Bootstrapping Architecture, Version 9.2.0*, 2009.
4. 3rd Generation Partnership Project (3GPP). *Access to network application functions using Hypertext Transfer Protocol over Transport Layer Security (HTTPS), Version 9.0.0*, 2009.

5. 3rd Generation Partnership Project (3GPP). *Bootstrapping interface (Ub) and network application function interface (Ua); Protocol details, Version 9.0.0*, 2009.
6. 3rd Generation Partnership Project (3GPP). *Numbering, Addressing and Identification, Version 9.2.0*, 2009.
7. M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in TLS. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pages 35–45, New York, NY, USA, 2006. ACM.
8. R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 581–590, New York, NY, USA, 2006. ACM.
9. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.
10. J. Engler, C. Karlof, E. Shi, and D. Song. Is it too late for PAKE? In *W2SP '09: Proceedings of the Web 2.0 Security and Privacy Workshop*, Oakland, California, USA, May 2009.
11. P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005.
12. A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol—Version 3.0. Internet Draft, Transport Layer Security Working Group, November 1996.
13. S. Gajek, L. Liao, and J. Schwenk. Stronger TLS bindings for SAML assertions and SAML artifacts. In *SWS '08: Proceedings of the 5th ACM CCS Workshop on Secure Web Services*, pages 11–20, New York, NY, USA, 2008. ACM.
14. S. Holtmanns, V. Niemi, P. Ginzboorg, P. Laitinen, and N. Asokan. *Cellular Authentication for Mobile and Internet Services.* John Wiley and Sons, 2008.
15. International Organization for Standardization, Genève, Switzerland. *ISO/IEC 19772:2009 Information technology—Security techniques—Authenticated encryption*, Feburary 2009.
16. International Organization for Standardization, Genève, Switzerland. *ISO/IEC 9798-4:1999/Cor 1:2009 Information technology—Security techniques—Entity authentication—Part 4: Mechanisms using a cryptographic check function*, September 2009.
17. F. Kohlar, J. Schwenk, M. Jensen, and S. Gajek. On cryptographically secure bindings of SAML assertions to TLS sessions. In *ARES '10: Proceedings of the 5th International Conference on Availability, Reliability and Security*, pages 62–69, Los Alamitos, CA, USA, 2010. IEEE Computer Society.
18. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997.
19. J. Lopez, R. Oppliger, and G. Pernul. Why have public key infrastructures failed so far? *Internet Research*, 15:554–556, 2005.
20. J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *USENIX '98: Proceedings of the 7th USENIX Security Symposium*, pages 201–216, San Antonio, Texas, USA, 1998. USENIX Association.
21. National Institute of Standards and Technology, Federal Information Processing Standards (FIPS) Publication 180-3. *Secure Hash Standard (SHS)*, October 2008.
22. V. Niemi and K. Nyberg. *UMTS Security.* John Wiley and Sons, 2003.
23. R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication or how to effectively thwart the man-in-the-middle. *Computer Communications*, 29(12):2238–2246, 2006.
24. R. Oppliger, R. Hauser, and D. Basin. SSL/TLS session-aware user authentication revisited. *Computers and Security*, 27(3-4):64–70, 2008.

25. R. Oppliger, R. Hauser, D. Basin, A. Rodenhaeuser, and B. Kaiser. A proof of concept implementation of SSL/TLS session-aware user authentication (TLS-SA). In W. Brauer, T. Braun, G. Carle, and B. Stiller, editors, *Kommunikation in Verteilten Systemen (KiVS)*, Informatik aktuell, pages 225–236. Springer Berlin Heidelberg, 2007.
26. L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):332–351, 1999.
27. M. Steiner, P. Buhler, T. Eirich, and M. Waidner. Secure password-based cipher suite for TLS. *ACM Transactions on Information and System Security (TISSEC)*, 4(2):134–157, 2001.
28. D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) protocol for TLS authentication. RFC 5054 (Informational), November 2007.