On the Security of the Pre-Shared Key Ciphersuites of TLS

Yong Li[1], Sven Schäge[2⋆], Zheng Yang[1], Florian Kohlar[1], Jörg Schwenk[1]

[1] Ruhr-Universität Bochum, Germany
{yong.li,zheng.yang,florian.kohlar,joerg.schwenk}@rub.de
[2] University College London, UK
s.schage@ucl.ac.uk

Abstract. TLS is by far the most important protocol on the Internet for negotiating secure session keys and providing authentication. Only very recently, the standard ciphersuites of TLS have been shown to provide provably secure guarantees under a new notion called Authenticated and Confidential Channel Establishment (ACCE) introduced by Jager *et al.* at CRYPTO'12. In this work, we analyse the variants of TLS that make use of pre-shared keys (TLS-PSK). In various environments, TLS-PSK is an interesting alternative for remote authentication between servers and constrained clients like smart cards, for example for mobile phone authentication, EMV-based payment transactions or authentication via electronic ID cards. First, we introduce a new and strong definition of ACCE security that covers protocols with pre-shared keys. Next, we prove that all ciphersuite families of TLS-PSK meet our strong notion of ACCE security. Our results do not rely on random oracles nor on any non-standard assumption.

Keywords: TLS, TLS-PSK, ACCE, Pre-Shared Keys, Authenticated Key Exchange, Secure Channels

1 Introduction

TLS is undeniably the most prominent key exchange protocol in use today. While the security of most web applications relies on the classical Diffie-Hellman and RSA-based ciphersuites of TLS, there also exist several important applications that make use of one of the less common ciphersuites [31,1,29]. One such application is (remote) authentication of resource-restricted clients like smart-cards. In these scenarios, computational efficiency and low power consumption often are one of the most important system features. Instead of using the public-key based ciphersuites of TLS, applications can apply a variant of TLS that assumes pre-shared symmetric keys between client and server. The corresponding ciphersuite

⋆ Supported by EPSRC grant number EP/J009520/1.

family is termed TLS with pre-shared keys (TLS-PSK) and available in many TLS releases and libraries, for example [28,24,7].

Related Work: On the Security of TLS. Since the introduction of its predecessor SSL, the security of TLS has often been the focus of security researchers and attackers worldwide. Over the time, several attacks on TLS have been published. Most of these attacks do not directly attack the cryptographic core of TLS, but rather exploit side-channels or vulnerabilities in associated technologies, like the famous Bleichenbacher attack [6], or attacks on the domain name system or the public-key infrastructure [20,10,26]. However, despite that no serious attacks on the cryptographic core of the current TLS protocol are known, determining exactly what security guarantees TLS provides has been an elusive problem for many years. This is partly due to the fact that the popular TLS ciphersuites provably do not provide security in the sense of authenticated key exchange (AKE) protocols [3], the classical and very strong standard notion of security of key exchange protocols (which requires that the session key remains indistinguishable from random even if the adversary obtains the communication transcript of the session). Until recently only security analyses of modified versions of TLS were published [18,16,27]. At CRYPTO 2012, Jager, Kohlar, Schäge, and Schwenk (JKSS) [17] were the first to present a detailed security analysis of the *unmodified* version of one of TLS's ciphersuite families. They showed that the cryptographic core of ephemeral Diffie-Hellman with mutual authentication is a provably secure authenticated and confidential channel establishment (ACCE) protocol in the standard model. ACCE is a new security notion that is particularly well suited to capture what protocols like TLS intuitively want to achieve: the establishment of a secure channel between client and server. Among its features, it not only formalizes confidentiality and integrity of messages exchanged between client and server, but also covers replay and re-ordering attacks. Very recently, Krawczyk, Paterson, and Wee (KPW) [22] and independently Kohlar, Schäge, Schwenk (KSS) [19] presented, while relying on different cryptographic assumptions and security models [3], extensions of the JKSS result to the remaining ciphersuite families. In particular, they show that TLS-RSA and TLS-DH also constitute ACCE protocols when used for mutual authentication setting and that TLS-RSA, TLS-DH, and TLS-DHE are ACCE secure in the practically important setting

[3] The security models and complexity assumptions differ mainly with respect to the capabilities granted to the adversary when corrupting and registering new parties and the application of the random oracle model.

of server-only authentication (for which they provide new formal security definitions).

Unfortunately, all previous results on the (ACCE) security of TLS are based on either i) new, non-standard security assumption like the PRF-ODH assumption introduced in [17] and refined in [22,19] or ii) strong idealizations such as the modeling of TLS's key derivation function as a random oracle [2] or assuming that the public-key encryption scheme in TLS-RSA is replaced with an IND-CCA secure one. Looking somewhat ahead, for the TLS-PSK ciphersuites, fortunately the situation is different, i.e. security can be based on standard assumptions only.

TLS with Pre-Shared Keys. The original specifications of the TLS protocol [11,12,13] do not explicitly include ciphersuites that support authentication and key exchange using pre-shared keys. However, since 2005 there exists an extension called "Pre-Shared Key Ciphersuites for Transport Layer Security" (TLS-PSK) which specifically describes such ciphersuites in RFC 4279 [14]. The TLS-PSK standard specifies three ciphersuites, `TLS_PSK`, `TLS_RSA_PSK` and `TLS_DHE_PSK`, each of which derives the master secret in a different way. In `TLS_PSK`, the master secret is solely based on the secret pre-shared keys. In the remaining ciphersuites the computation of the master secret is additionally dependent on freshly exchanged secrets via encrypted key transport in `TLS_RSA_PSK` or Diffie-Hellman key exchange in `TLS_DHE_PSK`. The intuition is that as long as either the pre-shared key or the freshly exchanged secret is not compromised, then the protocol produces a secure application key. All three ciphersuites assume that the client only has a pre-shared key for authentication. Although it is not as widespread as TLS with RSA key transport, several interesting and important scenarios for TLS with pre-shared keys exist where its efficiency makes TLS-PSK a much more attractive alternative than, for example, TLS with self-signed certificates.

– Since November 2010, the new electronic German ID (eID) card supports online remote authentication of the eID card holder to some online service (eService). Here TLS-PSK is applied to perform mutual authentication between the two parties [15].
– As a second example, we mention the application of TLS-PSK in the Generic Authentication Architecture, the 3GGP mobile phone standard for UMTS and LTE. According to ETSI TR 133 919 V11.0.0 (2012-11), TLS-PSK can be used to secure the communication between server and user equipment.

– An IETF draft from 2009 for EMV smart cards describes an authentication protocol based on TLS-PSK [29]. EMV chips are widely deployed and are used commonly for secure payment transactions [9].

CONTRIBUTION. In this paper, we provide a security analysis of all three TLS-PSK ciphersuites. Similar to classical TLS, it is provably impossible to show that the keys produced by TLS-PSK are indistinguishable from random. Therefore, as one of our main contributions, we introduce the first definition of ACCE security for authentication protocols with pre-shared keys. We do not propose a separate model but rather an extension of the ACCE model of JKSS to also cover authentication via pre-shared keys. Next, we introduce a strengthened variant of this definition called *asymmetric perfect forward secrecy*, that captures that protocol sessions of ACCE protocols with pre-shared keys may retain a strong level of confidentiality even if the long-term secrets of the client are exposed after the protocol run. Asymmetric perfect forward secrecy is a strong security notion that can hold for protocols that do not fulfill the standard notion of perfect forward secrecy. This allows us to prove the security of such protocols in a stronger security model than was previously possible. We show that `TLS_PSK` is ACCE secure (without forward secrecy), `TLS_RSA_PSK` is ACCE secure with asymmetric perfect forward secrecy and `TLS_DHE_PSK` is secure with (classical) perfect forward secrecy. Informally, our results say that TLS-PSK guarantees confidentiality and integrity of all messages exchange between client and server, unless the adversary has learned the pre-shared key or corrupted one of the parties to learn the application/session key. In `TLS_DHE_PSK` the communication remains confidential even if the adversary corrupts the pre-shared secret later on. In contrast, in `TLS_RSA_PSK` the communication remains confidential even if the adversary manages to corrupt the pre-shared key or the server's long-term key later on, but not both of them.

DOUBLE PRFS AND FORWARD SECRECY. To prove `TLS_RSA_PSK` and `TLS_DHE_PSK`, we introduce a variant of pseudo-random functions (PRFs), called double pseudo-random function (DPRF). Roughly, a DPRF takes as input two keys only one of which is generated randomly and kept secret from the attacker (as in classical PRFs). However, when the adversary makes its queries, not only the message but also the other key can entirely be specified by the adversary. Our notion of DPRF nicely abstracts the crucial mechanism in TLS-PSK that is required to guarantee (asymmetric) perfect forward secrecy. In our security proofs, we assume that TLS's key derivation function provides a suitable DPRF in the standard model.

Existing results on the security of HMAC support this assumption for TLS 1.1 when the pre-shared key has a specific bit length. Our new DPRF notion may be of independent interest beyond the scope of this work.

Note also, that for the TLS_PSK and TLS_DHE_PSK ciphersuites we neither have to rely on non-standard assumptions like the PRF-ODH assumption of JKSS to give a proof nor on idealized setup assumptions like the random oracle model. We can show that TLS_RSA_PSK is secure under our basic notion of ACCE security without any assumption on the public key encryption system used in TLS. This is because under the basic ACCE definition security can be derived solely from secrecy of the pre-shared keys. However, if we want to prove the ACCE security of TLS_RSA_PSK with asymmetric perfect forward secrecy in the *standard model* we need to assume that the public key encryption scheme is IND-CCA secure[4], similar to [22,19]. Thus, we do not consider TLS-RSA with RSA-PKCS encryption as it is currently used in practice. We remark that [22] were also able to prove security of the classical TLS ciphersuites based on RSA key transport with RSA-PKCS encryption in the random oracle model. It would be interesting to show that the results of KPW on TLS-RSA can be transferred to show that TLS-PSK with RSA-PKCS based key transport provides asymmetric perfect forward secrecy in the random oracle model.

LIMITATIONS. In our work, we give a dedicated security analysis for TLS-PSK. We believe that it is possible to give a more modularized analysis, similar to KPW [22] who analyzed the classical ciphersuites of TLS by abstracting the handshake phase into a Constrained-CCA-secure (CCCA) KEM that is combined with a secure authenticated encryption scheme. The benefit of the KPW analysis is re-usability: once the security proof is established for a generic CCCA-secure KEM, all that remains is to show that each of the ciphersuites indeed provides such a KEM.

2 Security Assumptions

To state our results, we will rely on standard security definitions for the Decisional Diffie-Hellman assumption (DDH), collision-resistant cryptographic hash functions, IND-CCA secure public key encryption schemes, (plain) pseudo-random functions (PRF), and stateful length-hiding authenticated encryption (sLHAE) schemes as recently defined in [30]. However, we will sometimes also rely on a new class of PRFs called double pseudo-random functions.

[4] KPW call this TLS-CCA.

Double Pseudo-Random Functions. Double pseudo-random functions can be thought of as a class of pseudo-random functions with two keys. Let $\mathsf{DPRF} : \mathcal{K}_{\mathsf{DPRF}_1} \times \mathcal{K}_{\mathsf{DPRF}_2} \times \mathcal{M}_{\mathsf{DPRF}} \rightarrow \mathcal{R}_{\mathsf{DPRF}}$ denote a family of deterministic functions, where $\mathcal{K}_{\mathsf{DPRF}_1}, \mathcal{K}_{\mathsf{DPRF}_2}$ is the key space, $\mathcal{M}_{\mathsf{DPRF}}$ is the domain and $\mathcal{R}_{\mathsf{DPRF}}$ is the range of PRF.

Intuitively, security requires that the output of the DPRF is indistinguishable from random as long as one key remains hidden from the adversary even if the adversary is able to adaptively specify the second key and the input message. To formalize security we consider the following security game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. Let $\mathsf{RF}_{\mathsf{DPRF}}(\cdot, \cdot)$ denote an oracle implemented by $\mathcal{C}$, which takes as input a key $k_j \in \mathcal{K}_{\mathsf{DPRF}_j}$ (where j is specified by the adversary via an Init query) and message $m \in \mathcal{M}_{\mathsf{DPRF}}$ and outputs a random value $z \in \mathcal{R}_{\mathsf{DPRF}}$.

1. The adversary first runs $\mathsf{Init}(j)$ with $j \in \{1, 2\}$ to specify the key $k_j \in \mathcal{K}_{\mathsf{DPRF}_j}$ that he wants to manipulate.
2. The challenger $\mathcal{C}$ samples $\hat{b} \xleftarrow{\$} \{0, 1\}$, and sets $u = (j \mod 2) + 1$. If $\hat{b} = 0$, the challenger samples $k_u \in_R \mathcal{K}_{\mathsf{DPRF}_u}$ and assigns $\mathsf{RF}_{\mathsf{DPRF}}(\cdot, \cdot)$ to either $\mathsf{DPRF}(\cdot, k_2, \cdot)$ or $\mathsf{DPRF}(k_1, \cdot, \cdot)$ depending on the value of u. For instance, if $u = 2$ then the random function $\mathsf{RF}_{\mathsf{DPRF}}$ is assigned to $\mathsf{DPRF}(\cdot, k_2, \cdot)$, and the $\mathcal{A}$ is allowed to specify k_1 arbitrarily in each query. If $\hat{b} = 1$, the challenger assigns $\mathsf{RF}_{\mathsf{DPRF}}$ to $\mathsf{RF}(\cdot, \cdot)$ which is a truly random function that takes as input key k_j and message m and outputs a value in the same range $\mathcal{R}_{\mathsf{DPRF}}$ as $\mathsf{DPRF}(\cdot, \cdot, \cdot)$.
3. The adversary may adaptively make queries $k_{j,i}$, m_i for $1 \leq i \leq q$ to oracle $\mathsf{RF}_{\mathsf{DPRF}}$ and receives the result of $\mathsf{RF}_{\mathsf{DPRF}}(k_{j,i}, m_i)$, where $k_{j,i}$ denotes the i-th key k_j chosen by $\mathcal{A}$.
4. Finally, $\mathcal{A}$ outputs its guess $\hat{b}' \in \{0, 1\}$ of $\hat{b}$. If $\hat{b} = \hat{b}'$, $\mathcal{A}$ wins.

Definition 1. *We say that* DPRF *is a* (t, ϵ)-secure *double pseudo-random function, if any adversary running in time t has at most an advantage of* ϵ *to distinguish the* DPRF *from a truly random function, i.e.*

$$\Pr\left[\hat{b} = \hat{b}'\right] \leq 1/2 + \epsilon.$$

The number of allowed queries q is upper bounded by t.

3 A Brief Introduction to TLS-PSK

This section describes the three sets of ciphersuites specified in TLS-PSK: `TLS_PSK`, `TLS_RSA_PSK` and `TLS_DHE_PSK`. In each of these ciphersuites, the master secret is computed using pre-shared keys which are symmetric keys shared in advance among the communicating parties. The main differences are in the way the master secret is computed. The following description is valid for *all* `TLS_PSK` versions. We only describe the cryptographically relevant messages and only those that deviate from the classical TLS ciphersuites. A detailed description can be found in the full version.

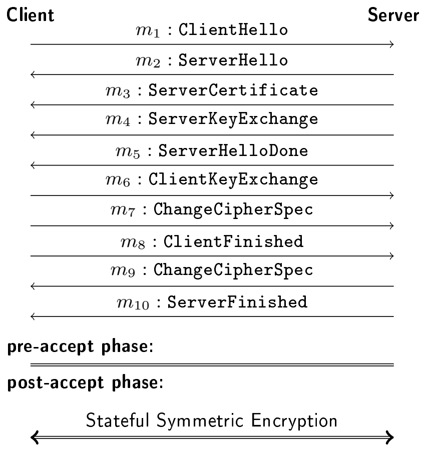

Fig. 1: Handshake in TLS-PSK

ServerCertificate. For `TLS_PSK` and `TLS_DHE_PSK`, the message is not included. In `TLS_RSA_PSK` cert_S contains a public key that is bound to the server's identity.

ServerKeyExchange. Since clients and servers may have pre-shared keys with many different parties, in the `ServerKeyExchange` message m_4, the server provides a `PSK identity hint` pointing to the PSK used for authentication. However, for ephemeral Diffie-Hellman key exchange, the Diffie-Hellman (DH) key exchange parameters are also contained in the `ServerKeyExchange` messages including information on the DH group (e.g. a large prime number $p \in \{0,1\}^{poly(\kappa)}$, where κ is the security parameter, and a generator $\langle g \rangle$ for a prime-order q subgroup of $\mathbb{Z}_p^*$), and the DH share T_S ($T_S = g^{t_S}$, where t_S is a random value in $\mathbb{Z}_q$). (We implictly assume that the client checks whether the received parameters are valid, in particular if T_S is indeed in the group generated by g.)

ClientKeyExchange. Message m_6 is called `ClientKeyExchange`. We describe the contents of this message for the ciphersuites `TLS_DHE_PSK`, `TLS_PSK` and `TLS_RSA_PSK` separately:

– For `TLS_PSK`, the message is not included.
– For ephemeral Diffie-Hellman key exchange `TLS_DHE_PSK`, it contains the Diffie-Hellman share T_C of the client, i.e. $T_C = g^{t_C}$.
– For the RSA-based key exchange `TLS_RSA_PSK` the client selects a 46-byte random value R and sends a 2-byte version number V and the

46-byte random value R encrypted under the server's RSA public key to the server.

Also, the client sends an identifier for the pre-shared key it is going to use when communicating with the server. This information is called `PSK identity`.

Computing The Master Secret. According to the original specification, released as RFC 4279 [14], the key derivation function of TLS, denoted here as $\mathsf{PRF}_{\mathsf{TLS}}$, is used when constructing the master secret. $\mathsf{PRF}_{\mathsf{TLS}}$ takes as input a secret, a seed, and an identifying label and produces an output of arbitrary length. We first describe the generic computation of the master secret *ms* for all ciphersuites using pre-shared keys. Then, a detailed description of all cases (`TLS_PSK`, `TLS_DHE_PSK`, and `TLS_RSA_PSK`) is provided. The *master secret ms* is computed as follows:

$$ms := \mathsf{PRF}_{\mathsf{TLS}}(pms, label_1 || r_C || r_S) \tag{1}$$

– `TLS_PSK` case: For `TLS_PSK`, the client/server is able to compute the *master secret ms* using the pre-master secret *pms*, from which all further secret values are derived. If the PSK is N bytes long, the *pms* consists of the 2-byte representation (uint16) of the integer value N, N zero bytes, the 2-byte representation of N once again, and the PSK itself, i.e. $pms := N||0...0||N||\mathsf{PSK}$. Since the first half of *pms* is constant for any PSK we get for `TLS_PSK` that the entire security of $\mathsf{PRF}_{\mathsf{TLS}}$ only relies on the second half of *pms*.
– `TLS_DHE_PSK` case: Let Z be the value produced for DH-based ciphersuites, i.e. $Z = g^{t_S t_C} = T_C^{t_S} = T_S^{t_C}$. The *pms* consists of a concatenation of four values: the uint16 len_Z indicating the length of Z, Z itself, the uint16 len_{PSK} showing the length of the PSK, and the PSK itself: $pms := len_Z||Z||len_{PSK}||\mathsf{PSK}$.
– `TLS_RSA_PSK` case: First, the pre-master secret concatenates the uint16 constant $\mathsf{C} = 48$, the 2-byte version number V, a 46-byte random value R, the uint16 len_{PSK} containing the length of the PSK, and the PSK itself, i.e. $pms := \mathsf{C}||\mathsf{V}||\mathsf{R}||len_{PSK}||\mathsf{PSK}$.

3.1 On the Security of $\mathsf{PRF}_{\mathsf{TLS}}$

In our security proof of `TLS_PSK`, we assume that the pseudo-random function of TLS ($\mathsf{PRF}_{\mathsf{TLS}}$) that is used for the computation of the master-secret constitutes a secure PRF in the standard model when applied with *pms* as the key. However to prove (asymmetric) perfect forward secrecy in TLS_DHE_PSK and TLS_RSA_PSK, *we assume that* $\mathsf{PRF}_{\mathsf{TLS}}$ *constitutes a secure DPRF (in the standard model)* where the key space of the DPRF consists of the key space of the pre-shared key and the key space of the freshly generated RSA or Diffie-Hellman secret. Unfortunately, existing results do not *directly* prove that $\mathsf{PRF}_{\mathsf{TLS}}$ as used in TLS-PSK is a secure DPRF. Nevertheless, they might in some cases serve as a strong indicator of the security of $\mathsf{PRF}_{\mathsf{TLS}}$. We provide a more detailed analysis of the plausibility of this assumption in the full version.

4 ACCE protocols

In this section, we present an extension of the formal security model for two party authenticated and confidential channel establishment (ACCE) protocols introduced by JKSS [17] to also cover scenarios with pre-shared, symmetric keys. Additionally, we extend the model to also address PKI-related attacks that exploit that the adversary does not have to prove knowledge of the secret key when registering a new public key [5]. (In [25] such attacks are generally called strong-key substitution attacks.) For better comparison with JKSS we will subsequently use boxes to highlight state variables that are essentially new in our model.

In this model, while emulating the real-world capabilities of an active adversary, we provide an 'execution environment' for adversaries following the tradition of the seminal work of Bellare and Rogaway [3] and its extensions [4,8,21,23,17]. Let $\mathcal{K}_0 = \{0,1\}^\kappa$ be the key space of the session key and $\mathcal{K}_1 = \{0,1\}^\kappa$ be the key space of the pre-shared keys.

Execution Environment. In the following let $\ell, d \in \mathbb{N}$ be positive integers. In the execution environment, we fix a set of ℓ honest parties $\{P_1, \ldots, P_\ell\}$. Each party is either identified by index i in the security experiment or a unique, fixed-length string id_i (which might appear in the protocol flows).

To cover authentication with symmetric keys, we extend the state of each party to also include pre-shared keys. Each party holds (symmetric) pre-shared keys with all other parties. We denote with $\mathsf{PSK}_{i,j} = \mathsf{PSK}_{j,i}$ the symmetric key shared between parties P_i and P_j. Each party P_i with $i \in \{1, \ldots, \ell\}$ also has access to a long-term public/private key pair (pk_i, sk_i). Formally, each party maintains the state variables given in Table 1.

The first two variables, sk_i and PSK_i, are used to store keys that are used in the protocol execution while the remaining variables are solely used to define security. (When defining security the latter are additionally

Variable	Description
sk_i	stores the secret key of a public key pair (pk_i, sk_i)
$\boxed{\mathsf{PSK}_i}$	a vector which contains an entry $\mathsf{PSK}_{i,j}$ per party P_j
τ_i	denotes, that sk_i was corrupted after the τ_i-th query of $\mathcal{A}$
$\boxed{f_i}$	a vector denoting the freshness of all pre-shared keys,
	containing one entry $f_{i,j} \in \{\mathsf{exposed}, \mathsf{fresh}\}$ for each entry in PSK_i

Table 1: Internal States of Parties

managed and updated by the challenger.) The variables of each party P_i will be initialized according to the following rules:

- The long-term key pair (pk_i, sk_i) and pre-shared key vector PSK_i are chosen randomly from the key space. For all parties P_i, P_j with $i, j \in \{1, \ldots, \ell\}$ and with $i \neq j$, and pre-shared keys PSK_i it holds that $\mathsf{PSK}_{i,j} = \mathsf{PSK}_{j,i}$ and $\mathsf{PSK}_{i,i} := \emptyset$.
- All entries in f_i are set to fresh.
- τ_i is set to $\tau_i := \infty$, which means that all parties are initially not corrupted.

In the following, we will call party P_i uncorrupted iff $\tau_i = \infty$. Thus, we do not consider a dedicated variable that holds the corruption state of the secret key sk_i. Each honest party P_i can sequentially and concurrently execute the protocol multiple times. This is modeled by a collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. Oracle π_i^s behaves as party P_i carrying out a process to execute the s-th protocol instance with some partner P_j (which is determined during the protocol execution). All oracles of P_i have access to the long-term keys sk_i and PSK_i with $j \in \{1, \ldots, \ell\}$. Moreover, we assume each oracle π_i^s maintains a list of independent internal state variables with the semantics given in Table 2. The variables Φ_i^s, Pid_i^s, ρ_i^s,

Variable	Description
Φ_i^s	denotes π_i^s's execution-state in $\{\mathtt{negotiating}, \mathtt{accept}, \mathtt{reject}\}$
Pid_i^s	stores the identity of the intended communication partner
ρ_i^s	denotes the role $\rho_i^s \in \{\mathsf{Client}, \mathsf{Server}\}$
$\mathsf{K}_i^s = (k_{\mathsf{enc}}, k_{\mathsf{dec}})$	stores the application keys K_i^s
$\mathsf{St}_i^s = (u, v, st_e, st_d, C)$	stores the current states of the sLHAE scheme
T_i^s	records the transcript of messages sent and received by π_i^s
$\boxed{\mathsf{kst}_i^s}$	denotes the freshness $\mathsf{kst}_i^s \in \{\mathsf{exposed}, \mathsf{fresh}\}$ of the session key
b_i^s	stores a bit $b \in \{0, 1\}$ used to define security

Table 2: Internal States of Oracles

K_i^s, st_e, st_d, and T_i^s are used by the oracles to execute the protocol. The remaining variables are only used to define security. The variables of each oracle π_i^s will be initialized by the following rules:

- The execution-state Φ_i^s is set to `negotiating`.
- The variable kst_i^s is set to fresh.
- The bit b_i^s is chosen at random.
- The counters u, v are initialized to 0.
- All other variables are set to only contain the empty string $\emptyset$.

At some point, each oracle π_i^s completes the execution with a decision state $\Phi_i^s \in \{\texttt{accept}, \texttt{reject}\}$. Furthermore, we will always assume (for simplicity) that $\mathsf{K}_i^s = \emptyset$ if an oracle has not reached `accept`-state (yet).

Matching Conversations. To formalize the notion that two oracles engage in an on-line communication, we define partnership via *matching conversations* as proposed by Bellare and Rogaway [3]. We use the variant by JKSS.

Definition 2. *We say that an oracle π_i^s has a* matching conversation *to oracle π_j^t, if*

- *π_i^s has sent all protocol messages and T_j^t is a prefix of T_i^s, or*
- *π_j^t has sent all protocol messages and $\mathsf{T}_i^s = \mathsf{T}_j^t$.*

To keep our definition of ACCE protocols general we do not consider protocol-specific definitions of partnership like for example [22] who define partnership of TLS sessions using only the first three messages exchanged in the handshake phase.

Adversarial Model. An adversary $\mathcal{A}$ in our model is a PPT taking as input the security parameter 1^κ and the public information (e.g. generic description of above environment), which may interact with these oracles by issuing the following queries.

$\mathsf{Send}^{\mathsf{pre}}(\pi_i^s, m)$**:** This query sends message m to oracle π_i^s. The oracle will respond with the next message m^* (if there is any) that should be sent according to the protocol specification and its internal states.
After answering a $\mathsf{Send}^{\mathsf{pre}}$ query, the variables $(\Phi_i^s, \mathsf{Pid}_i^s, \rho_i^s, \mathsf{K}_i^s, T_i^s)$ will be updated depending on the protocol specification. This query is essentially defined as in JKSS.

$\mathsf{RegisterParty}(\mu, pk_\mu, [psk])$**:** This query allows $\mathcal{A}$ to register a new party with a new identity μ and a static public key (pk_μ) to be used for party

P_μ. In response, if the same identity μ is already registered (either via a $\mathsf{RegisterParty}$-query or $\mu \in [\ell]$), a failure symbol $\perp$ is returned. Otherwise, a new party P_μ is added with the static public key pk_μ. The secret key sk_μ is set to a constant. The parties registered by this query are considered corrupted and controlled by the adversary. If $\mathsf{RegisterParty}$ is the τ'-th query of the adversary, P_μ is initialized with $\tau_\mu = \tau'$. If the adversary also provides a pre-shared key psk, then this key will be implemented for every party P_i with $i \in [\ell]$ as key $\mathsf{PSK}_{i,\mu}$.[5] Otherwise, the simulator chooses a random key $psk \xleftarrow{\$} \{0,1\}^\kappa$ and sets $\mathsf{PSK}_{i,\mu} = \mathsf{PSK}_{\mu,i} := psk$ for all parties P_i before outputting psk. The corresponding entries $f_{i,\mu}$ in the vectors of the other parties P_i with $i \in [\ell]$ are set to $\mathsf{exposed}$. Via this query we extend the ACCE model of JKSS to also model key registration.

$\mathsf{RevealKey}(\pi_i^s)$**:** Oracle π_i^s responds to a $\mathsf{RevealKey}$-query with the contents of variable K_i^s, the application keys. At the same time the challenger sets $\mathsf{kst}_i^s = \mathsf{exposed}$. If at the point when $\mathcal{A}$ issues this query there exists another oracle π_j^t having matching conversation to π_i^s, then we also set $\mathsf{kst}_j^t = \mathsf{exposed}$ for π_j^t. This query slightly deviates from JKSS.[6].

$\mathsf{Corrupt}(P_i, [P_j])$**:** Depending on the second input parameter, oracle π_i^1 responds with certain long-term secrets of party P_i. This query extends the corruption capabilities of JKSS to symmetric keys.

- If $\mathcal{A}$ queries $\mathsf{Corrupt}(P_i)$ or $\mathsf{Corrupt}(P_i, \emptyset)$[7], oracle π_i^1 returns the long-term secret key sk_i of party P_i. If this query is the τ-th query issued by $\mathcal{A}$, then we say that P_i is τ-corrupted and π_i^1 sets $\tau_i := \tau$.
- If $\mathcal{A}$ queries $\mathsf{Corrupt}(P_i, P_j)$, oracle π_i^1 returns the symmetric pre-shared key $\mathsf{PSK}_{i,j}$ stored in PSK_i and sets $f_{i,j} := \mathsf{exposed}$.
- If $\mathcal{A}$ queries $\mathsf{Corrupt}(P_i, \top)$, oracle π_i^1 returns the vector PSK_i and sets $f_{i,j} := \mathsf{exposed}$ for all entries $f_{i,*} \in f_i$.

$\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)$**:** This query takes as input two messages m_0 and m_1, length parameter len, and header data H. If $\Phi_i^s \neq \mathtt{accept}$ then π_i^s returns $\perp$. Otherwise, it proceeds as depicted in Figure 2, depending on the random bit $b_i^s \xleftarrow{\$} \{0,1\}$ sampled by π_i^s at the beginning of the

[5] This is just for simplicity. Modeling different pre-shared keys between the registered party and every other party is equivalent to registering multiple parties with a single shared key each.

[6] JKSS implicitly located the specification of when to set $\mathsf{kst}_j^t = \mathsf{exposed}$ into the security definition.

[7] The party P_i is not adversarially controlled.

game and the internal state variables of π_i^s. This query is essentially defined as in JKSS.

$\mathsf{Decrypt}(\pi_i^s, C, H)$**:** This query takes as input a ciphertext C and header data H. If π_i^s has $\Phi_i^s \neq$ '`accept`' then π_i^s returns $\perp$. Otherwise, it proceeds as depicted in Figure 2. This query is essentially defined as in JKSS.

```
$\underline{\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)}$:
$u := u + 1$
$(C^{(0)}, st_e^{(0)}) \overset{\$}{\leftarrow} \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m_0, st_e)$
$(C^{(1)}, st_e^{(1)}) \overset{\$}{\leftarrow} \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m_1, st_e)$
If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$
$(C_u, H_u, st_e) := (C^{(b)}, H, st_e^{(b)})$
Return $C_u$
```

```
$\underline{\mathsf{Decrypt}(\pi_i^s, C, H)}$:
$v := v + 1$
If $b_i^s = 0$, then return $\perp$
$(m, st_d) = \mathsf{StE.Dec}(k_{\mathsf{dec}}^{\rho}, H, C, st_d)$
If $v > u$ or $C \neq C_v$ or $H \neq H_v$,
  then $\mathsf{phase} := 1$
If $\mathsf{phase} = 1$ then return $m$
```

Here $u, v, b_i^s, \rho, k_{\mathsf{enc}}^{\rho}, k_{\mathsf{dec}}^{\rho}, C$ denote the values stored in the internal variables of π_i^s.

Fig. 2: Encrypt and Decrypt oracles in the ACCE security experiment.

Definition 3 (Correctness). *We say that an* ACCE *protocol Π is correct, if for any two oracles π_i^s, π_j^t that have matching conversations with $\mathsf{Pid}_i^s = j$ and $\mathsf{Pid}_j^t = i$ and $\Phi_i^s =$* `accept` *and $\Phi_j^t =$* `accept` *it always holds that $\mathsf{K}_i^s = \mathsf{K}_j^t$.*

Secure ACCE Protocols. We define security via an experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Security Game. Assume there is a global variable *pinfo* which stores the role information of each party for the considered protocol Π.[8] In the game, the following steps are performed:

1. Given the security parameter κ, $\mathcal{C}$ implements the collection of oracles $\{\pi_i^s : i, j \in [\ell], s \in [d]\}$ with respect to Π and *pinfo*. In this process, $\mathcal{C}$ generates long-term keys PSK_i for all parties $i \in [\ell]$. Next it additionally generates long-term key pairs (pk_i, sk_i) for all parties $i \in [\ell]$ that require them (e.g. if the corresponding party is a server in the `TLS_RSA_PSK` protocol). Finally, $\mathcal{C}$ gives all identifiers $\{id_i\}$, all public keys (if any), and *pinfo* to $\mathcal{A}$.

[8] This information is simply used to determine which party also holds asymmetric key pairs besides the shared symmetric keys.

2. Next the adversary may start issuing $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{RevealKey}$, $\mathsf{Corrupt}$, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$, and $\mathsf{RegisterParty}$ queries.
3. At the end of the game, the adversary outputs a triple (i, s, b') and terminates.

In the following, we provide a general security definition for ACCE protocols. It will subsequently be referred to when formalizing specific definitions for ACCE protocols that provide no forward secrecy, perfect forward secrecy or asymmetric perfect forward secrecy. We have tried to keep the details of the execution environment and the definition of security close to that of JKSS. Intuitively, our security definition mainly differs from JKSS in that it considers adversaries that also have access to the new $\mathsf{RegisterParty}$ query and the extended $\mathsf{Corrupt}$ query.

Definition 4 (ACCE Security). *We say that an adversary (t, ϵ)-*breaks *an ACCE protocol, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:*

1. *When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s such that*
   - *π_i^s 'accepts' with $\mathsf{Pid}_i^s = j$ when $\mathcal{A}$ issues its τ_0-th query, and*
   - *both P_i and the intended partner P_j*[9] *are not corrupted throughout the security game and*
   - *π_i^s has internal state $\mathsf{kst}_i^s = \mathsf{fresh}$, and*
   - *there is no unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t.*

   If an oracle π_i^s accepts in the above sense, then we say that π_i^s accepts maliciously*.*
2. *When $\mathcal{A}$ terminates and outputs a triple (i, s, b') such that*
   - *π_i^s 'accepts' – with a unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t – when $\mathcal{A}$ issues its τ_0-th query, and*
   - *$\mathcal{A}$ did not issue a $\mathsf{RevealKey}$-query to oracle π_i^s nor to π_j^t, i.e. $\mathsf{kst}_i^s = \mathsf{fresh}$, and*
   - *P_i is τ_i-corrupted and P_j is τ_j-corrupted,*

   then the probability that b' equals b_i^s is bounded by

   $$\left|\Pr[b_i^s = b'] - 1/2\right| \geq \epsilon.$$

   If adversary $\mathcal{A}$ outputs (i, s, b') with $b' = b_i^s$ and the above conditions are met, we say that $\mathcal{A}$ answers the encryption-challenge correctly*.*

[9] The party P_j is not adversarially corrupted, i.e. $j \in [\ell]$. This means that P_j has not been registered by a $\mathsf{RegisterParty}$ query. Otherwise $\mathcal{A}$ may obtain all corresponding secure keys and trivially make oracle π_i^s accept.

We say that the ACCE protocol is (t, ϵ)-secure*, if there exists no adversary that* (t, ϵ)-breaks *it.*

Let us now define security more concretely. We consider three levels of forward secrecy. We start with a basic security definition for protocols that do not provide any form of forward secrecy.

Definition 5 (ACCE Security without Forward Secrecy). *We say that an ACCE protocol is (t, ϵ)-secure* without forward secrecy (NoFS)*, if it is (t, ϵ)-secure with respect to Definition 4 and $\tau_i = \tau_j = \infty$.*

The next definition considers PFS in the classical sense for both, client and server, as in JKSS.

Definition 6 (ACCE Security with Perfect Forward Secrecy). *We say that an ACCE protocol is (t, ϵ)-secure* with perfect forward secrecy (PFS)*, if it is (t, ϵ)-secure with respect to Definition 4 and $\tau_i, \tau_j \geq \tau_0$.*

In the following, we provide our new definition of asymmetric perfect forward secrecy which is similar to that of classical perfect forward secrecy except that only the client is allowed to be corrupted after it has accepted.

Definition 7 (ACCE Security with Asymmetric Perfect Forward Secrecy). *We say that an ACCE protocol is (t, ϵ)-secure* with asymmetric perfect forward secrecy (APFS)*, if it is (t, ϵ)-secure with respect to Definition 4 and it holds that $\tau_i = \infty$ and $\tau_j \geq \tau_0$ if π_i^s has internal state $\rho = \mathsf{Server}$ or $\tau_i \geq \tau_0$ and $\tau_j = \infty$ if π_i^s has internal state $\rho = \mathsf{Client}$.*

5 Security Analysis of Pre-Shared Key Ciphersuites for Transport Layer Security

In this section, we present our results for each of the TLS-PSK ciphersuites. Due to space restrictions, the proofs are given in the full version.

Theorem 1. *Let μ be the output length of $\mathsf{PRF_{TLS}}$ and let λ be the length of the nonces. Assume that $\mathsf{PRF_{TLS}}$ is a $(t, \epsilon_{\mathsf{PRF}})$-secure PRF when keyed with the pre-master secret $pms := N||0...0||N||\mathsf{PSK}$ or the master secret ms. Suppose the hash function H is $(t, \epsilon_{\mathsf{H}})$-secure, and the sLHAE scheme is $(t, \epsilon_{\mathsf{StE}})$-secure. Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the* `TLS_PSK` *protocol in the sense of Definition 5 with $t \approx t'$ it holds that*

$$\epsilon_{\mathsf{tls}} \leq (d\ell)^2 \left(\frac{1}{2^{\lambda-1}} + 3\epsilon_{\mathsf{DPRF}} + 3\epsilon_{\mathsf{PRF}} + 2\epsilon_{\mathsf{H}} + \frac{1}{2^{\mu-1}} + 6\epsilon_{\mathsf{StE}} \right).$$

Theorem 2. *Let μ be the output length of $\mathsf{PRF}_{\mathsf{TLS}}$ and let λ be the length of the nonces. Assume that $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{DPRF}})$-secure DPRF when keyed with the pre-master secret $pms := len_Z||Z||len_{PSK}||\mathsf{PSK}$ (that consists of the pre-shared secret PSK and the Diffie-Hellman value Z). Assume that $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{PRF}})$-secure PRF when keyed with the master secret ms. Suppose the hash function H is $(t, \epsilon_{\mathsf{H}})$-secure, the DDH-problem is $(t, \epsilon_{\mathsf{DDH}})$-hard in the group G used to compute Z, and the sLHAE scheme is $(t, \epsilon_{\mathsf{StE}})$-secure. Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the `TLS_DHE_PSK` protocol in the sense of Definition 6 with $t \approx t'$ we get*

$$\epsilon_{\mathsf{tls}} \leq (d\ell)^2 \left(\frac{1}{2^{\lambda-1}} + 3\epsilon_{\mathsf{DPRF}} + 3\epsilon_{\mathsf{PRF}} + 2\epsilon_{\mathsf{H}} + \frac{1}{2^{\mu-1}} + \epsilon_{\mathsf{DDH}} + 6\epsilon_{\mathsf{StE}} \right).$$

Theorem 3. *Let μ be the output length of $\mathsf{PRF}_{\mathsf{TLS}}$ and let λ be the length of the nonces. Assume that $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{DPRF}})$-secure DPRF when keyed with the pre-master secret $pms := \mathsf{C}||\mathsf{V}||\mathsf{R}||len_{PSK}||\mathsf{PSK}$ (that consists of the pre-shared key PSK and the random key R that is exchanged between client and server). Assume that $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{PRF}})$-secure PRF when keyed with the master secret ms. Suppose the hash function H is $(t, \epsilon_{\mathsf{H}})$-secure, the public key encryption scheme PKE is $(t, \epsilon_{\mathsf{PKE}})$-secure (IND-CCA). Suppose that the sLHAE scheme is $(t, \epsilon_{\mathsf{StE}})$-secure. Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the `TLS_RSA_PSK` protocol (where the key transport mechanism is implemented via PKE) in the sense of Definition 7 with $t \approx t'$ it holds that*

$$\epsilon_{\mathsf{tls}} \leq (d\ell)^2 \left(\frac{1}{2^{\lambda-1}} + \epsilon_{\mathsf{PKE}} + 3\epsilon_{\mathsf{DPRF}} + 3\epsilon_{\mathsf{PRF}} + 2\epsilon_{\mathsf{H}} + \frac{1}{2^{\mu-1}} + 6\epsilon_{\mathsf{StE}} \right).$$

Technical Overview of the Security Proofs. At a high level, the security proofs are similar to that of JKSS. From a technical standpoint, the security proof of `TLS_PSK` is simpler than that of the classical ciphersuites of TLS as security only relies on the secrecy of the pre-shared secrets. Roughly, in the proofs of the classical TLS ciphersuites one additionally has to establish that the key exchange mechanism produces a shared secret in the first place. To prove `TLS_RSA_PSK` and `TLS_DHE_PSK` we exploit the DPRF-security of $\mathsf{PRF}_{\mathsf{TLS}}$. The challenge is to show that the master secret is indistinguishable from random although the adversary may reveal the pre-shared secret or a freshly generated ephemeral secret. Intuitively, if only one of these values remains unrevealed by the adversary, then at least one input key to the DPRF $\mathsf{PRF}_{\mathsf{TLS}}$ is (indistinguishable from) random. Therefore, $\mathsf{PRF}_{\mathsf{TLS}}$ computes a random-looking master secret which in turn can be used to derive secure application keys.

Acknowledgements. We would like to thank Kenny Paterson and the anonymous referees for their valuable comments and suggestions.

References

1. Mohamad Badra and Pascal Urien. Toward SSL integration in SIM smartcards. In *WCNC*, pages 889–893. IEEE, 2004.
2. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.
3. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, August 1994.
4. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In Michael Darnell, editor, *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer, December 1997.
5. Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the Station-to-Station (STS) protocol. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 1999.
6. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, August 1998.
7. BouncyCastle Software Developers. Bouncy Castle Crypto APIs, 2013. `http://www.bouncycastle.org/`.
8. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, May 2001.
9. Chunhua Chen, Shaohua Tang, and Chris J. Mitchell. Building general-purpose security services on EMV payment cards. In Angelos D. Keromytis and Roberto Di Pietro, editors, *SecureComm*, volume 106 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 29–44. Springer, 2012.
10. Italo Dacosta, Mustaque Ahamad, and Patrick Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 2012.
11. T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746.
12. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746.
13. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878.

14. P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005.
15. German Federal Office for Information Security (BSI). TR-03112, Das eCard-API-Framework, 2005. `https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/tr03112/index_htm.html`.
16. Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec*, volume 5324 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2008.
17. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, August 2012.
18. Jakob Jonsson and Burton S. Kaliski Jr. On the security of RSA encryption in TLS. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 127–142. Springer, August 2002.
19. Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DH and TLS-RSA in the standard model. *IACR Cryptology ePrint Archive*, 2013:367, 2013.
20. Florian Kohlar, Jörg Schwenk, Meiko Jensen, and Sebastian Gajek. Secure bindings of SAML assertions to TLS sessions. In *ARES*, pages 62–69. IEEE Computer Society, 2010.
21. Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, August 2005.
22. Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448. Springer, 2013.
23. Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, November 2007.
24. Nikos Mavrogiannopoulos and Simon Josefsson. The GnuTLS Transport Layer Security library. Last updated 2013-03-22, `http://gnutls.org`.
25. Alfred Menezes and Nigel P. Smart. Security of signature schemes in a multi-user setting. *Des. Codes Cryptography*, 33(3):261–274, 2004.
26. Christopher Meyer and Jörg Schwenk. Lessons learned from previous SSL/TLS attacks - a brief chronology of attacks and weaknesses. *IACR Cryptology ePrint Archive*, 2013:49, 2013.
27. Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. The TLS handshake protocol: A modular analysis. *Journal of Cryptology*, 23(2):187–223, April 2010.
28. OpenSSL. The OpenSSL project, 2013. `http://www.openssl.org`.
29. L.Cogneau P. Urien and P. Martin. EMV support for TLS-PSK. draft-urien-tls-psk-emv-02, February 2011.
30. Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, December 2011.
31. Pascal Urien. Introducing TLS-PSK authentication for EMV devices. In Waleed W. Smari and William K. McQuay, editors, *CTS*, pages 371–377. IEEE, 2010.