On the Security of the Pre-Shared Key Ciphersuites of TLS[1]

Yong Li [2*], Sven Schäge [3†], Zheng Yang[*], Florian Kohlar[*] and Jörg Schwenk[*]

[*]Ruhr-Universität Bochum, Germany
{yong.li,zheng.yang,florian.kohlar,joerg.schwenk}@rub.de
[†]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 [49, 1, 42]. 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 family is termed TLS with pre-shared keys (TLS-PSK) and available in many TLS releases and libraries [41, 35, 10, 44, 45, 46, 48, 11].

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 [8], or attacks on the domain name system or the

[1] A preliminary version of this paper appears in the proceedings of PKC 2014. This is the full version.
[2] Supported by Secure eMobility grant number 01ME12025
[3] Supported by EPSRC grant number EP/J009520/1.

public-key infrastructure [29, 14, 38]. 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 classical sense of authenticated key exchange (AKE) protocols, the classical and very strong standard notion of security of key exchange protocols [36, 34, 47, 50, 26]. Until recently only security analyses of modified versions of TLS were published [27, 23, 39]. At CRYPTO 2012, Jager, Kohlar, Schäge, and Schwenk (JKSS) [25] 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) [32] and independently Kohlar, Schäge, Schwenk [28] presented, while relying on different cryptographic assumptions and security models [1], 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 and that TLS-RSA, TLS-DH, and TLS-DHE are ACCE secure in the practically important setting of server-only authentication (for which they provide new formal security definitions). When proving security in the standard model, both KPW and KSS assume that the public key encryption system used for key exchange in TLS-RSA is IND-CCA secure. However, KPW also gave a proof of security in the random oracle model where the public key encryption is only required to be OW-PCA.

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 [25] and refined in [32, 28] or ii) strong idealizations such as the modeling of TLS's key derivation function as a random oracle [3] or assuming that the public-key encryption scheme in TLS-RSA is substituted with a IND-CCA secure one. Looking somewhat ahead, for the TLS ciphersuites with pre-shared keys, 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 [15, 16, 17] 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 [19]. (Yet another extension termed "TLS Pre-Shared Key (PSK) Ciphersuites with NULL Encryption" proposes variants of the TLS-PSK ciphersuites that can be used only for authentication, i.e. when channel encryption is disabled [9].) 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 Handshake layer yields 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

[1] 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.

with RSA key transport, several interesting and important scenarios for TLS with pre-shared keys exist.

- Since November 2010, the new electronic German ID (eID) card supports online remote authentication of the eID card holder to some online service (eService). The most important network channel involved in the eID online authentication mechanism is a TLS channel, where the eID card shares symmetric keys with the authentication endpoints. Here TLS-PSK is applied to perform mutual authentication between the card holder and some online service. A technical report by the German Federal Office for Information Security (BSI) [21] describes in detail how the pre-shared key known to the eID-Server and the eService is bound to the TLS channel.

- 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 (e.g. handheld telephone or laptop with a mobile broadband adapter).

- An IETF draft from 2009 for EMV smart cards describes an authentication protocol based on TLS-PSK [42]. EMV chips are widely deployed and are used commonly for secure payment transactions [13]. The draft describes how the identity information and pre-shared keys stored on the EMV chip can be used to establish a TLS-PSK channel.

The main advantage of TLS-PSK over the standard ciphersuites (with self-signed certificates) is that it avoids computationally expensive public key operations. This particularly pays off in systems with energy-constrained devices like mobile phones or mobile payment stations. Often, in such systems the end points are initialized with pre-shared keys in some secure environment. At the same time, the network layout is hierarchical such that clients technically only directly communicate with a trusted central server that in turn routes their messages to the intended communication partner. This keeps the size of the key material that has to be stored in the clients small and virtually static. Its efficiency makes TLS-PSK a much more attractive alternative in these scenarios than, for example, TLS with self-signed certificates.

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. We provide an overview of our results in Figure 1.

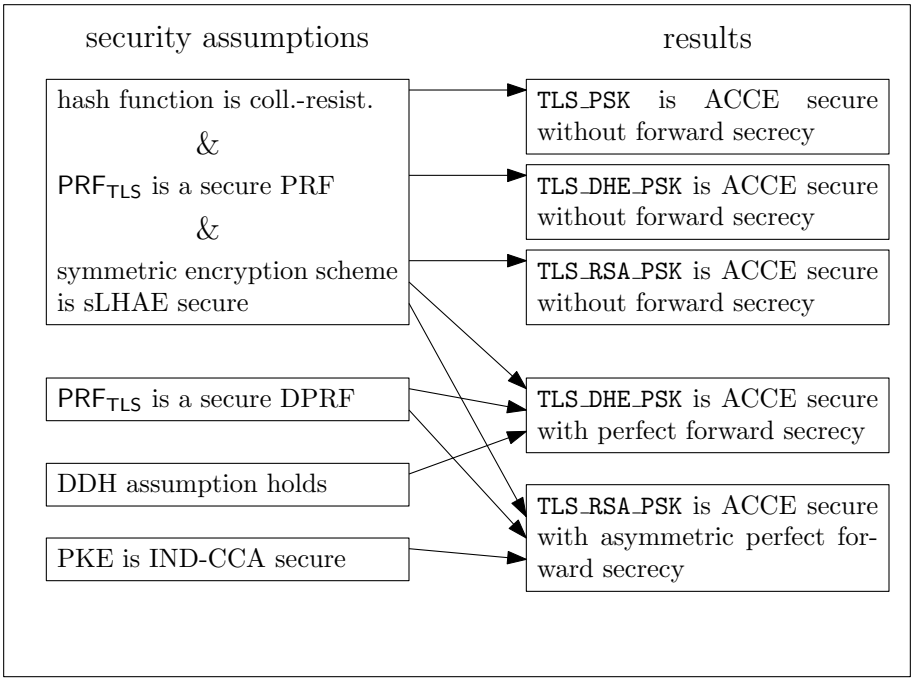

Figure 1: Summary of Results

Double PRFs and Perfect Forward Secrecy. To prove our results on 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 directly support this assumption for TLS 1.1 when the pre-shared key has a specific bit length. We believe that our new DPRF notion, can 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. 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, similar to [32, 28]. We remark that [32] were also able to prove security of the classical TLS ciphersuites based on RSA key transport 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 [32] who analyzed the classical ciphersuites of TLS by abstracting the handshake phase into a Constrained-CCA-secure (CCCA) key encapsulation mechanism that is combined with a secure authenticated encryption scheme. The benefit of the KPW analysis is re-usability: once the security proof of TLS is established for a generic CCCA secure KEM, all that remains is to show that each of the ciphersuites indeed provides such a key encapsulation mechanism.

We emphasize once again that `TLS_RSA_PSK` can be shown to be ACCE secure (without forward secrecy) without imposing any assumptions on the public key system used for the transport of the ephemeral key. This is because under the basic ACCE definition security can be derived solely from secrecy of the pre-shared keys. We stress that when showing that `TLS_RSA_PSK` provides asymmetric perfect forward secrecy, we do not consider TLS-RSA with RSA-PKCS encryption as it is currently used in practice. Instead we rather assume that TLS uses a generic IND-CCA secure encryption scheme that is secure in the standard model. It would be interesting to show that the results of KPW on TLS-RSA with RSA-PKCS encryption can be transferred to show that TLS-PSK with RSA-PKCS based key transport provides asymmetric perfect forward secrecy in the random oracle model.

2 Preliminaries

In the following we will briefly recall the definitions and primitives our analysis relies on. We denote with $\emptyset$ the empty string, and with $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ the set of integers between 1 and n. We use $\oplus$ to denote XOR. If A is a set, then $a \xleftarrow{\$} A$ denotes the action of sampling a uniformly random element from A. If A is a probabilistic algorithm, then $a \xleftarrow{\$} A$ denotes that A is run with fresh random coins and returns a. In the following, κ will denote the security parameter.

2.1 The Decisional Diffie-Hellman Assumption

Let G be a group of prime order q. Let g be a generator of G. Given (g, g^a, g^b, g^c) for $a, b, c \in \mathbb{Z}_q$ the decisional Diffie-Hellman (DDH) assumption says that it is hard to decide whether $c = ab \bmod q$.

Definition 1. We say that the DDH problem is $(t, \epsilon_{\mathsf{DDH}})$-*hard* in G, if for all adversaries $\mathcal{A}$ that run in time t it holds that for $a, b, z \xleftarrow{\$} \mathbb{Z}_q$

$$\left|\Pr\left[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1\right] - \Pr\left[\mathcal{A}(g, g^a, g^b, g^z) = 1\right]\right| \leq \epsilon_{\mathsf{DDH}},$$

where the probability is over the random coins of $\mathcal{A}$ and the random choices of a, b, z.

2.2 Collision-Resistant Hash Function

A *collision-resistant hash function* is a deterministic algorithm Hash which given a key $k \in \mathcal{K}_{\mathsf{Hash}}$ (with $\log(|\mathcal{K}_{\mathsf{Hash}}|)$ polynomial in κ) and a bit string m outputs a hash value $w = \mathsf{Hash}(k, x)$ in the hash space $\{0,1\}^\chi$ (with χ polynomial in κ). If k is clear from the context we write $\mathsf{Hash}(\cdot)$ short for $\mathsf{Hash}(k, \cdot)$.

Definition 2. We say that Hash is a (t, ϵ)-*secure* collision-resistant hash function, if any t-time adversary $\mathcal{A}$ that is given $k \xleftarrow{\$} \mathcal{K}_{\mathsf{Hash}}$ has an advantage of at most ϵ to compute two inputs m, m' with $m \neq m'$ and $\mathsf{Hash}(m) = \mathsf{Hash}(m')$.

2.3 Public Key Schemes

A public key encryption (PKE) scheme consists of three polynomial time algorithms PKE =($\mathsf{PKE.Gen}$, $\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$) with the following semantics:

- $(pk, sk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$: is a probabilistic polynomial-time key generation algorithm which generates a (public) encryption key pk and a secret decryption key sk on input of the security parameter κ.
- $C \xleftarrow{\$} \mathsf{PKE.Enc}(pk, m)$: is a probabilistic polynomial-time encryption algorithm which takes as inputs a public key pk and a message m, outputs ciphertext $C \in \mathcal{C}$, where $\mathcal{C}$ is a ciphertext space.
- $m \leftarrow \mathsf{PKE.Dec}(sk, C)$: is a deterministic polynomial-time decryption algorithm which takes as input a key sk and a ciphertext $C \in \mathcal{C}$, and outputs either a message $m \in \mathcal{M}$, where $\mathcal{M}$ is a message space, or an error symbol $\perp$.

The (IND-CCA) security of a PKE scheme PKE = ($\mathsf{PKE.Gen}$, $\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$) via the following game that is played between a challenger and an adversary.

1. The challenger computes $(pk, sk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$ and gives pk to the adversary.
2. The adversary may adaptively decrypt polynomially (in κ) many ciphertexts C of his choice.
3. At one point the adversary sends a message m_0 to the challenger.
4. The challenger samples $b \xleftarrow{\$} \{0, 1\}$. Then he draws a uniformly random message m_1 from the message space. Next it computes $C^* \xleftarrow{\$} \mathsf{PKE.Enc}(pk, m_b)$ and sends C^* to the adversary.
5. The adversary may adaptively decrypt polynomially (in κ) many ciphertexts C of his choice with the restriction that C^* is not among the values queried by the adversary.
6. The adversary outputs his guess $b' \in \{0, 1\}$ of b. If $b = b'$ the adversary wins.

We assume that the overall number of queries made to the challenger is $q = q(\kappa)$. We call $Pr\left[b = b'\right]$ the success probability of the adversary in winning the above game. We call $\left|\Pr\left[b = b'\right] - 1/2\right|$ the advantage of the adversary.

Definition 3 (PKE Security)**.** We say that PKE is a (t, ϵ)-secure PKE scheme, if an adversary running in time t has an advantage of at most ϵ, i.e.

$$\Pr\left[b = b'\right] \leq 1/2 + \epsilon,$$

while the number of allowed queries q is upper bounded by t.

2.4 Stateful Length-Hiding Authenticated Encryption

We use the stateful variant of LHAE security (sLHAE) as originally defined by Paterson *et al.* [43] and used by JKSS [25] for their analysis of TLS-DHE. Paterson *et al.* showed that the CBC-based record layer encryption of TLS meets the notion of sLHAE under suitable assumptions on the used cryptographic components and the implementation. A *stateful symmetric encryption scheme* basically consists of two algorithms $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$.

- $(C, st'_e) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m, st_e)$: takes as input a secret key $k \in \mathcal{K}_{sLHAE}$, an output ciphertext length $\mathsf{len} \in \mathbb{N}$, some header data $H \in \{0,1\}^*$, a plaintext $m \in \mathcal{M}_{sLHAE}$, and the current state $st_e \in \{0,1\}^*$, and outputs either a ciphertext $C \in \{0,1\}^{\mathsf{len}}$ and an updated state st'_e or an error symbol $\perp$ if for instance the output length len is not valid for the message m.
- $(m', st'_d) = \mathsf{StE.Dec}(k, H, C, st_d)$: takes as input a key k, header data H, a ciphertext C, and the current state $st_d \in \{0,1\}^*$, and returns an updated state st'_d and a value m' which is either the message encrypted in C, or a distinguished error symbol $\perp$ indicating that C is not a valid ciphertext.

Both encryption state st_e and decryption state st_d are initialized to the empty string $\emptyset$. Algorithm $\mathsf{StE.Enc}$ may be probabilistic, while $\mathsf{StE.Dec}$ is always deterministic. See [43] for more details.

The security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. Figure 2 describes how the oracles ENC and DEC respond to $\mathcal{A}$'s queries. The values u, v and phase are all initialized to 0 at the beginning of the security game.

1. The challenger $\mathcal{C}$ selects $b \xleftarrow{\$} \{0,1\}$ and $k \xleftarrow{\$} \{0,1\}^\kappa$ and sets $st_e := \emptyset$ and $st_d := \emptyset$,
2. $\mathcal{A}$ may adaptively query the encryption oracle ENC q_{ENC} times and the decryption oracle DEC q_{DEC} times.
3. Finally, $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$.

$\underline{\mathsf{ENC}(m_0, m_1, \mathsf{len}, H)}$:	$\underline{\mathsf{DEC}(C, H)}$:
$u := u + 1$	$v := v + 1$
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_0, st_e)$	If $b = 0$, then return $\perp$
$(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_1, st_e)$	$(m, st_d) = \mathsf{StE.Dec}(k, H, C, st_d)$
If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$	If $v > u$ or $C \neq C_v$ or $H \neq H_v$, then $\mathsf{phase} := 1$
$(C_u, H_u, st_e) := (C^{(b)}, H, st_e^{(b)})$	If $\mathsf{phase} = 1$ then return m
Return C_u	Return $\perp$

Figure 2: $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ oracles for the stateful LHAE security experiment

Definition 4. We say that the stateful symmetric encryption scheme $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$ is (t, ϵ)-*secure* , if any adversary running in time t has an advantage of at most ϵ to output b' such that $b' = b$, i.e.

$$\Pr\left[b' = b\right] \leq 1/2 + \epsilon,$$

while the number of allowed queries q is upper bounded by t.

2.5 Plain Pseudo-Random Functions.

Let $\mathsf{PRF} : \mathcal{K}_{\mathsf{PRF}} \times \mathcal{M}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}}$ denote a family of deterministic functions, where $\mathcal{K}_{\mathsf{PRF}}$ is the key space, $\mathcal{M}_{\mathsf{PRF}}$ is the domain and $\mathcal{R}_{\mathsf{PRF}}$ is the range of PRF.

To define security, we consider the following security game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. Let $\pi_{\mathsf{PRF}}(\cdot)$ denote an oracle implemented by $\mathcal{C}$, which takes as input a message $m \in \mathcal{M}_{\mathsf{PRF}}$ and outputs a value $z \in \mathcal{R}_{\mathsf{PRF}}$.

1. The challenger samples $b \xleftarrow{\$} \{0,1\}$. If b = 0, the challenger samples $k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$ and assigns oracle $\pi_{\mathsf{PRF}}(\cdot)$ to $\mathsf{PRF}(k,\cdot)$. If b = 1, the challenger assigns oracle $\pi_{\mathsf{PRF}}(\cdot)$ to $\mathsf{RF}(\cdot)$ which is a truly random function associated with the same range and domain as $\mathsf{PRF}(k,\cdot)$.

2. The adversary may adaptively make queries x_i for $1 \leq i \leq q$ to oracle $\pi_{\mathsf{PRF}}(\cdot)$ and receives the result depending on the random bit b.

3. Finally, the adversary outputs its guess $b' \in \{0,1\}$ of b. If $b = b'$ the adversary wins.

We assume that the overall number of queries made to the challenger is $q = q(\kappa)$. As before, we call $Pr\left[b = b'\right]$ the success probability of the adversary in winning the above game. We call $|\Pr\left[b = b'\right] - 1/2|$ the advantage of the adversary.

Definition 5. We say that PRF is a (t,ϵ)-*secure* pseudo-random function, if any adversary running in time t has an advantage of at most ϵ to distinguish the PRF from a truly random function, i.e.

$$\Pr\left[b = b'\right] \leq 1/2 + \epsilon,$$

while the number of allowed queries q is upper bounded by t.

2.6 Double Pseudo-Random Functions

Double pseudo-random functions can be thought of a class of PRFs with two keys. Let $\mathsf{DPRF} : \mathcal{K}_{\mathsf{DPRF}_1} \times \mathcal{K}_{\mathsf{DPRF}_2} \times \mathcal{M}_{\mathsf{DPRF}} \to \mathcal{R}_{\mathsf{DPRF}}$ 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 through 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, the adversary outputs its guess $\hat{b}' \in \{0,1\}$ of $\hat{b}$. If $\hat{b} = \hat{b}'$ the adversary wins.

As before, we let $\Pr\left[\hat{b} = \hat{b}'\right]$ denote the success probability of the adversary and $\left|\Pr\left[\hat{b} = \hat{b}'\right] - 1/2\right|$ its advantage.

Definition 6. We say that DPRF is a (t, ϵ)-*secure* pseudo-random function, if any adversary running in probabilistic polynomial time t has at most an advantage of ϵ to distinguish the DPRF from a truly random function, i.e.

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

Again the number of allowed queries q is upper bounded by t.

Observe that any (t, ϵ)-secure DPRF trivially gives rise to a plain PRF: if the DPRF is secure after adaptive message queries – even when one of the keys can be specified by the adversary – it remains of course secure when both keys are chosen at random and kept secret from the adversary. However, such a function can be viewed as a PRF where the key space consist of all possible pairs of keys (k_1, k_2). Also, if the DPRF can generate output values which are indistinguishable from random if the adversary adaptively specifies keys and messages, it remains secure if the adversary is only allowed to specify messages. The following lemma holds for any message space and output space.

Lemma 1. *Suppose that* $\mathsf{DPRF}(k_1, k_2, m)$ *is a* $(t, \epsilon_{\mathsf{PRF}})$*-secure DPRF with key spaces* $\mathcal{K}_{\mathsf{DPRF}_1}$ *and* $\mathcal{K}_{\mathsf{DPRF}_2}$ *according to Definition 6. Then* $\mathsf{DPRF}(k_1, k_2, m)$ *is a* $(t, \epsilon_{\mathsf{PRF}})$*-secure PRF for key space* $\mathcal{K}_{\mathsf{DPRF}_1}$*, key space* $\mathcal{K}_{\mathsf{DPRF}_2}$*, or* $\mathcal{K}_{\mathsf{DPRF}_1} \times \mathcal{K}_{\mathsf{DPRF}_2}$*.*

Proof. To show that $\mathsf{DPRF}(k_1, k_2, m)$ is a $(t, \epsilon_{\mathsf{PRF}})$-secure PRF for key space $\mathcal{K}_{\mathsf{DPRF}_1}$ (or key space $\mathcal{K}_{\mathsf{DPRF}_2}$) we can imagine a PRF simulator that simply queries $\mathsf{Init}(0)$ (or $\mathsf{Init}(1)$) to its DPRF challenger and subsequently only relays the message queries and responses. To show that $\mathsf{DPRF}(k_1, k_2, m)$ is a $(t, \epsilon_{\mathsf{PRF}})$-secure PRF for key space $\mathcal{K}_{\mathsf{DPRF}_1} \times \mathcal{K}_{\mathsf{DPRF}_2}$ we use that it is a PRF for $\mathcal{K}_{\mathsf{DPRF}_1}$ and key space $\mathcal{K}_{\mathsf{DPRF}_2}$. So as long as either k_1 or k_2 is chosen uniformly at random $\mathsf{DPRF}(k_1, k_2, m)$ is a PRF even independent of the choice of the other key. However it of course remains a PRF if the other key is chosen uniformly as well. □

A Practical Construction of a DPRF from PRFs. There is a simple way to construct a DPRF from two PRFs. Assume we have two PRFs $\mathsf{PRF}(\cdot,\cdot) : \mathcal{K}_{\mathsf{PRF}} \times \mathcal{M}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}}$ and $\mathsf{PRF}'(\cdot,\cdot) : \mathcal{K}_{\mathsf{PRF}'} \times \mathcal{M}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}'}$. We can then construct a DPRF with $\mathcal{K}_{\mathsf{DPRF}_1} = \mathcal{K}_{\mathsf{PRF}}$ and $\mathcal{K}_{\mathsf{DPRF}_2} = \mathcal{K}_{\mathsf{PRF}'}$ and message space $\mathcal{M}_{\mathsf{DPRF}} = \mathcal{M}_{\mathsf{PRF}}$. On input $k_1 \in \mathcal{K}_{\mathsf{DPRF}_1}$ and $k_2 \in \mathcal{K}_{\mathsf{DPRF}_2}$ and message $m \in \mathcal{M}_{\mathsf{DPRF}}$, the DPRF proceeds as follows:

$$\mathsf{DPRF}(k_1, k_2, m) := \mathsf{PRF}(k_1, m) \oplus \mathsf{PRF}'(k_2, m).$$

Lemma 2. *Suppose that* PRF *and* PRF' *are* $(t, \epsilon_{\mathsf{PRF}})$*-secure pseudo-random functions according to Definition 5. Then the above* DPRF *is* $(t', \epsilon_{\mathsf{DPRF}})$*-secure according to Definition 6 with* $t \approx t'$ *and* $\epsilon_{\mathsf{DPRF}} \leq \epsilon_{\mathsf{PRF}}$*.*

Proof. Let S_δ denote the event that $b' = b$ in Game δ. Let $\mathsf{Adv}_\delta := |\Pr[\mathsf{S}_\delta] - 1/2|$ denote the advantage of $\mathcal{A}$ in Game $\delta$and $\Pr[\mathsf{S}_\delta]$ its success probability. Consider the following sequence of games.

Game 0. This game equals the DPRF security experiment. Thus, for some ϵ_{DPRF} we have

$$\Pr[\mathsf{S}_0] = 1/2 + \epsilon_{\mathsf{DPRF}}.$$

Game 1. Assume the adversary queries $\mathsf{Init}(j)$ with $j \in 1, 2$. Let $u = (j \mod 2) + 1$. In this game, we either change the function $\mathsf{PRF}(k_1^*, \cdot)$ to a truly random function $\mathsf{RF}(\cdot)$ (if $u = 1$) or the function $\mathsf{PRF}'(k_2^*, \cdot)$ (if $u = 2$). If there exists a polynomial time adversary $\mathcal{A}$ that can distinguish this game from the previous game, we can construct an algorithm $\mathcal{B}$ using $\mathcal{A}$ that breaks the security of PRF or PRF'. Exploiting the security of PRF (or PRF'), we have that

$$|\Pr[\mathsf{S}_0] - \Pr[\mathsf{S}_1]| \leq \epsilon_{\mathsf{PRF}}.$$

Since DPRF's output is computed as

$$\mathsf{DPRF}(k_1, k_2, m) := \mathsf{PRF}(k_1, m) \oplus \mathsf{PRF}'(k_2, m)$$

we also have that even when the adversary entirely and adaptively specifies either $\mathsf{PRF}(k_1, m)$ or $\mathsf{PRF}'(k_2, m)$ via the q queries granted in the DPRF game, the output $\mathsf{DPRF}(k_1, k_2, m)$ is still indistinguishable from random as the other PRF output $\mathsf{PRF}'(k_2, m)$ or $\mathsf{PRF}(k_1, m)$ remains random. Therefore, if only one PRF is exchanged with a truly random function, the entire function $\mathsf{DPRF}(k_1, k_2, m)$ behaves like a truly random function. We have

$$\Pr[\mathsf{S}_1] = 1/2 \Leftrightarrow \mathsf{Adv}_1 = 0.$$

To show that this game is indistinguishable from the previous one observe that $\mathcal{B}$ can simulate the $\mathsf{RF}_{\mathsf{DPRF}}(k_j, \cdot)$ queries made by $\mathcal{A}$ as $\pi_{\mathsf{PRF}} \oplus \mathsf{PRF}(k_j, \cdot)$ where π_{PRF} is the oracle in the PRF security experiment. If $\pi_{\mathsf{PRF}} = \mathsf{RF}(\cdot)$ then $\mathcal{B}$'s simulation yields a distribution that is equal to this game, otherwise it is equal to the previous game. Finally, $\mathcal{B}$ can simply forward $\mathcal{A}$'s response to its challenger in the PRF game.

Collecting probabilities from Game 0 to Game 1 yields Lemma 2. □

3 A Brief Introduction to Transport Layer Security

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. As sketched before, in `TLS_RSA_PSK` the computation of the master secret is additionally dependent on a random value produced by the client that is sent to the server encrypted with its public key. In `TLS_DHE_PSK` the master secret is computed using the pre-shared keys and a fresh Diffie-Hellman key that is exchanged between client and server. The following description is valid for *all* `TLS_PSK` versions. The TLS handshake protocol consists of 10 messages, whose content ranges from constant byte values to tuples of cryptographic values. Not all messages are relevant for our security proof, we list them merely for completeness. All messages are prepended with a numeric tag that identifies the type of message, a length value, and the version number of TLS. Also, all messages are sent through the 'TLS Record Layer', which at startup provides no encryption nor any other cryptographic transformations.

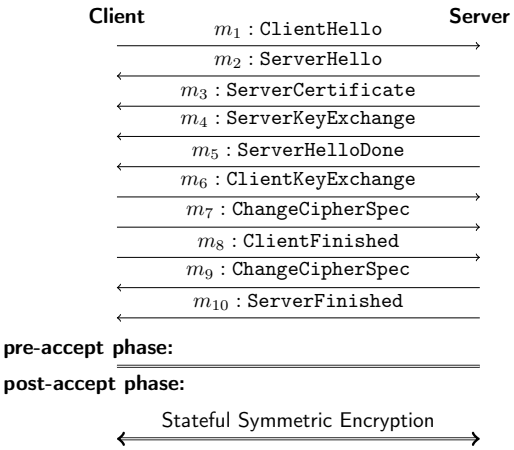

Figure 3: TLS handshake for the PSK key exchange algorithm and associated ciphersuites

ClientHello. Message m_1 is the `ClientHello` message. In this message, one or more TLS-PSK ciphersuites that are supported by the client are included. For our analysis the only important value is r_C, the random value chosen by the client. It consists of 32 bytes (256 Bits), where 4 Bytes are usually used to encode the local time of the client. The remaining 28 Bytes are chosen randomly by the client. If the client wants to resume a previous TLS session, he may optionally include a TLS *session ID* value received from the server in a previous session. (This value is not protected cryptographically and should thus not be confused with session IDs used in formal security proofs.) This is followed by a list `cs-list` of *ciphersuites*, where each ciphersuite is a tuple of key exchange method, signing, encryption and MAC algorithms, coded as two bytes. Data compression is possible before encryption and is signaled by the inclusion of zero or more compression methods.

ServerHello. The `ServerHello` message m_2 has the same structure as `ClientHello`. The TLS-server can select one of the PSK ciphersuites specified by the client and includes this ciphersuite in the `ServerHello` message. In our analysis the value r_S is important which is drawn randomly by the server.

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 TLS-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.)

ServerHelloDone. The `ServerHelloDone` message m_5 does not contain any data, but consists only of a constant tag with byte-value '14' and a length value '0'. The server sends this message in order to inform the client to proceed with the next phase of the protocol.

ClientKeyExchange. Message m_6 is called `ClientKeyExchange`. We describe the contents of this message for `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 send an identifier for the pre-shared key it is going to use when communicating with the server. This information is called `PSK identity`.

ChangeCipherSpec. To signal the 'start of encryption' to the server, the client sends message m_7 `ChangeCipherSpec` that simply contains the byte value '1' to the server.

ClientFinished. The next data to be sent is the `ClientFinished` message, m_8, which consists of an encryption C_C of fin_C concatenated with a MAC value. The messages C_C and fin_C are computed as follows: $fin_C := \mathsf{PRF}(ms, label_3||\mathsf{H}(m_1||\ldots||m_7))$ and $C_C := \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, len, H, fin_C, st_e)$. The application key $k_{\mathsf{enc}}^{\mathsf{Client}}$ and the master secret ms are described below.

ServerFinished. After the server has received messages m_8, the server can also compute pms, ms, the encryption and MAC keys, and the `ServerFinished` message fin_S. He can then decrypt m_8 and check fin_C by computing the pseudo-random value on the messages sent and received by the server. If this check fails, the server '$rejects$' and aborts the handshake. If the check is successful, he '$accepts$' and sends the message $m_9$containing C_S which is the encryption of fin_S to the client. The messages C_S and fin_S are computed as follows: $fin_S := \mathsf{PRF}(ms, label_4||\mathsf{H}(m_1||\ldots||m_9))$ and $C_S := \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, len, H, fin_S, st_e)$.

Computing The Master Secret. According to the original specification, released as RFC 4279 [19], 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` version, 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 and a 46-byte random value R, the uint16 len_{PSK} containing the length of the PSK, and the PSK itself, i.e. $pms := \mathsf{C}||\mathsf{V}||\mathsf{R}||len_{PSK}||\mathsf{PSK}$.

Computing The Application Keys. After computing the *master secret* ms, it is stored for the lifetime of the TLS session, and pms is erased from memory. The *master secret* ms is subsequently used, together with the two random nonces and another fixed $label_2$, to derive all encryption and MAC keys. The following four application keys (encryption and MAC keys for each direction) are computed by using $\mathsf{PRF}_{\mathsf{TLS}}$, where the inputs are now the master secret ms, $label_2$ and r_C, r_S. More precisely, the key material $k_{\mathsf{enc}}^{\mathsf{Client}} := (K_{\mathsf{enc}}^{C\to S}, K_{\mathsf{mac}}^{C\to S})$ and $k_{\mathsf{dec}}^{\mathsf{Client}} := (K_{\mathsf{enc}}^{S\to C}, K_{\mathsf{mac}}^{S\to C})$ is computed as

$$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := \mathsf{PRF}_{\mathsf{TLS}}(ms, label_2||r_C||r_S), \tag{2}$$

where $k_{\mathsf{enc}}^{\mathsf{Client}}$ is used to encrypt and authenticate data sent from the client to the server, and $k_{\mathsf{dec}}^{\mathsf{Client}}$ is used to decrypt and verify data received from the server.

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. In the following we will analyse the plausibility of these assumptions in the light of existing results. What considerably complicates our analysis is that TLS 1.1 and TLS 1.2 specify different implementations of $\mathsf{PRF}_{\mathsf{TLS}}$. We therefore start with a detailed description of $\mathsf{PRF}_{\mathsf{TLS}}$ in TLS 1.1 and TLS 1.2

Implementation of $\mathsf{PRF}_{\mathsf{TLS}}$ in TLS 1.1. In TLS 1.1, the output of the key derivation is computed as:

$$\mathsf{PRF}_{\mathsf{TLS}}(pms, m) = \text{HMAC\_MD5}'(pms1, m) \oplus \text{HMAC\_SHA}'(pms2, m) \tag{3}$$

where $pms1$ is the first half of the pre-master secret and $pms2$ is the second half, i.e. $pms = pms1||pms2$. As described in detail before, in TLS-PSK the input message m is derived from some constant, public label and the messages exchanged in the protocol so far (depending on

the ciphersuite used). HMAC_MD5′ is computed from several concatenations and iterations of HMAC_MD5 that all use the same input key $pms1$. Similarly, HMAC_SHA′ is computed from several concatenations and iterations of HMAC_SHA that again all use the same input key $pms2$. In general, the data expansion function HMAC_X′(k,m) for key k and message m is defined as

$$\text{HMAC\_X}'(k, m) = \text{HMAC\_X}(k, A(1)||m) \,||\, \text{HMAC\_X}(k, A(2)||m) \,||\, \ldots \tag{4}$$

where the $A(i)$ are defined as

$$A(0) = m, \tag{5}$$
$$A(i) = \text{HMAC\_X}(k, A(i-1)). \tag{6}$$

In the above, we use HMAC_X to refer to the standard HMAC algorithm that uses X as the underlying hash function [30]. TLS allows to generate arbitrary output lengths for HMAC_X′ (that are not necessarily multiples of the output length of HMAC_X). To this end, one simply computes an HMAC_X′ output that is (slightly) larger than the target length via Equation 4. Next, the last output bits of this result are just discarded (until we meet the target length).

Implementation of $\mathsf{PRF}_{\mathsf{TLS}}$ in TLS 1.2. The definition of $\mathsf{PRF}_{\mathsf{TLS}}$ in TLS 1.2 is different from TLS 1.1. First, the new standard allows client and server to negotiate the underlying hash function. However, SHA-256 is used in all pre-defined ciphersuites specified in the TLS standard and generally recommended. Second, the computation of $\mathsf{PRF}_{\mathsf{TLS}}$ no longer relies on two different hash functions but only on a single one. For SHA-256 the function thus simply looks like

$$\mathsf{PRF}_{\mathsf{TLS}}(pms, m) = \text{HMAC\_SHA-256}'(pms, m). \tag{7}$$

Existing Results on the Security of $\mathsf{PRF}_{\mathsf{TLS}}$. Let now us give a brief summary of the most important theoretical results. In [2], Bellare proved that HMAC is a pseudo-random function when the underlying compression function of the hash function is a PRF when keyed by either the data input or the chaining value. In 2008, Foque, Pointcheval, and Zimmer (FPZ) showed that, while relying on [2], for *any* key distribution with high min-entropy, HMAC [18] is a good strong randomness extractor under security assumptions that are related to the fact that the compression function of the underlying hash function behaves like a pseudo-random function [22]. In 2010, Fischlin, Lehmann, and Wagner (FLW) [20] specifically analyzed the key derivation function of TLS 1.1. In their analysis, FLW show that HMAC_X′ is a secure PRF if HMAC_X is a secure PRF. FLW rely on [2] (who showed that HMAC_X is pseudo-random) to base the security of $\mathsf{PRF}_{\mathsf{TLS}}$ on the security of the compression functions of the underlying hash functions. Very recently, Koblitz and Menezes gave a separation result showing that the proof of [2] actually does not apply to the standardized version of HMAC [30] without modifications (of the security assumptions). They also present a new proof of security of HMAC as standardized in [40] that holds in the uniform model of complexity. However, due to its large tightness loss, Koblitz and Menezes doubt that even their new and strengthened security proof is "good enough to serve as a convincing real-world guarantee of security of HMAC".

Application to $\mathsf{PRF}_{\mathsf{TLS}}$. Unfortunately, none of these results *directly* proves that $\mathsf{PRF}_{\mathsf{TLS}}$ as used in TLS-PSK behaves like a DPRF. Nevertheless, they might in some cases serve as a strong indicator of the security of $\mathsf{PRF}_{\mathsf{TLS}}$.

- When using TLS_PSK, the security of $\mathsf{PRF}_{\mathsf{TLS}}$ only relies on the secret key PSK, which is located in the last bits of pms. The results of FLW on the pseudo-randomness of HMAC_X′ do not only make it appear plausible that $\mathsf{PRF}_{\mathsf{TLS}}$ constitutes a PRF in TLS 1.1 but also in TLS 1.2.[2]

- As another important example, consider the security of $\mathsf{PRF}_{\mathsf{TLS}}$ as specified in TLS 1.1 when using TLS_DHE_PSK and TLS_RSA_PSK. Recall that in TLS_DHE_PSK the pre-master secret is computed as $pms := len_Z||Z||len_{PSK}||\mathsf{PSK}$. Assume that the length of PSK is such that $len_Z||Z$ and $len_{PSK}||\mathsf{PSK}$ have equal bit length. When splitting pms in two halves, we now get that $pms1 = len_Z||Z$ and $pms2 = len_{PSK}||\mathsf{PSK}$ are independent random keys. At this point, we may again rely on the results of FLW (and Lemma 2) to deduce that if an adversary may not reveal both, Z and PSK, the output of $\mathsf{PRF}_{\mathsf{TLS}}$ remains indistinguishable from a random value. In this case, $\mathsf{PRF}_{\mathsf{TLS}}$ practically constitutes a DPRF if HMAC_MD5 is a secure PRF for the key space $len_Z||Z$ (with random Z) and HMAC_SHA is a secure PRF with key space $len_{PSK}||\mathsf{PSK}$ (for random PSK). We obtain an analogous result for TLS_RSA_PSK if the length of PSK is such that $pms1 = \mathsf{C}||\mathsf{V}||\mathsf{R}$ and $pms2 = len_{PSK}||\mathsf{PSK}$.

However, in general PSK may have arbitrary size. In TLS_DHE_PSK for example, if PSK is relatively small it is likely that $pms2$ also consists of several bits of Z. In this case, HMAC_SHA′ is used with a key that not only consists of random bits of PSK but also of possibly adversarially manipulated bits of Z. Thus parts of the key bits of HMAC_SHA′ may be specified by the adversary. It is not clear if the results of FLW transfer to these situations as well.

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 [25] 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 [7]. (In [37] 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 [4] and its extensions [6, 12, 31, 33, 25]. 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 string id_i with fixed length (which might appear in the protocol flows).

Long-term keys. 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.

[2]Technically, the key spaces of HMAC_X′ need to be defined distinctly. In TLS 1.2, HMAC_X′ is required to be a PRF for the key space that consist of all $N||0...0||N||\mathsf{PSK}$ with random PSK while in TLS 1.1 the key space would consist of all $N||\mathsf{PSK}$.

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, all parties maintain several state variables as described in Table 2.

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

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, see below. (When defining security the latter are additionally 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 characterized 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 as described in Table 2.

Variable	Description
Φ_i^s	denotes the execution-state $\Phi_i^s \in \{\texttt{negotiating}, \texttt{accept}, \texttt{reject}\}$
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 oracle 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

The variables Φ_i^s, Pid_i^s, ρ_i^s, 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. As in JKSS, u, v are simple counters used for defining security, whereas st_e and st_d hold the state information of the symmetric encryption system. C represents a list of ciphertexts that can be indexed by u and v.

The variables of each oracle π_i^s will be initialized with 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 $\emptyset$.

revealed. We find our solution very natural. We stress that the problem is not restricted to our analysis of TLS-PSK but rather seems fundamental to security protocols in general (similar to the problem that an adversary may always drop the last protocol message which makes one party end up accepting although the transcripts of its partner oracle is actually different). We also remark that by using a distinct definition of partnership, we could seemingly avoid this problem, for example by using the definition of [32] (or an adapted form) that only spans the first three messages of TLS. However, this would come at the cost of generality of our definition and we refrain from doing so. Also, we remark that when providing a new partnership definition that is specific to some protocol, for example a truncated version of the transcript, there must be some additional formal evidence that this definition actually uniquely identifies sessions.

At some point, each oracle π_i^s completes the execution with a 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 [4]. We use the variant by JKSS. We assume that messages in a transcript T_i^s are represented as binary strings. Let $|\mathsf{T}_i^s|$ denote the number of messages. Assume there are two transcripts T_i^s and T_j^t of oracles π_i^s and π_j^t respectively. We say that T_i^s is a prefix of T_j^t and the first $|\mathsf{T}_i^s|$ messages in transcripts T_i^s and T_j^t are pairwise equivalent as binary strings.

Definition 7. 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 [32] who define partnership of TLS sessions using only the first three messages exchanged in the handshake phase (see Remark 1 below).

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 $\mathsf{PSK}_{i,\mu}$.[3] 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 to cover the attacks mentioned in Remark 1.[4].

$\mathsf{Corrupt}(P_i, [P_j])$**:** 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 $\mathsf{Corrupt}(P_i, \emptyset)$[5], 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

[3] 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.

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

[5] The party P_i is not adversarially controlled.

in Figure 4, depending on the random bit $b_i^s \xleftarrow{\$} \{0,1\}$ sampled by π_i^s at the beginning of the 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 4. This query is essentially defined as in JKSS.

$\underline{\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)}$:	$\underline{\mathsf{Decrypt}(\pi_i^s, C, H)}$:
$u := u + 1$	$v := v + 1$
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m_0, st_e)$	If $b_i^s = 0$, then return $\perp$
$(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m_1, st_e)$	$(m, st_d) = \mathsf{StE.Dec}(k_{\mathsf{dec}}^{\rho}, H, C, st_d)$
If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$	If $v > u$ or $C \neq C_v$ or $H \neq H_v$,
$(C_u, H_u, st_e) := (C^{(b)}, H, st_e^{(b)})$	then $\mathsf{phase} := 1$
Return C_u	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.

Figure 4: Encrypt and Decrypt oracles in the ACCE security experiment.

Remark 1. In the execution environment we need the state variable kst_i^s because we have to cope with a theoretical attack caused by the RevealKey query. Consider the situation when an uncorrupted server oracle π_i^s accepts and just sends out the encrypted finished message C_S to its partner oracle π_j^t which up to now has matching conversation (which is defined next) with π_i^s . However at this point an adversary $\mathcal{A}$ may reveal the session key of π_i^s, drop C_S and computes a ciphertext C'_S which encrypts the same finished message but differs from C_S. Next, $\mathcal{A}$ sends C'_S to π_j^t. This problem is that now π_j^t still accepts, although it does not have matching conversation to π_i^s. To thwart this attack, we modify the RevealKey query and the corresponding security definition as compared to JKSS. In our definition, the challenger in the security game simply keeps track of which session keys have been queried by the adversary via kst_i^s (and kst_j^t) and does not allow the adversary to break the security of any oracle whose session key has been revealed. We find our solution very natural. JKSS use a very similar solution. Roughly, they specify when a session key has been revealed by a partner oracle in the security definition whereas we use the definition of the RevealKey query to do so. We stress that the problem is not restricted to our analysis of TLS-PSK but rather seems fundamental to security protocols in general (similar to the problem that an adversary may always drop the last protocol message which makes one party end up accepting although the transcripts of its partner oracle is actually different). We also remark that by using a distinct definition of partnership, we could seemingly avoid this problem, for example by using the definition of [32] (or an adapted form) that only spans the first three messages of TLS. However, this would come at the cost of generality of our definition and we refrain from doing so. Also, we remark that when providing a new partnership definition that is specific to some protocol, for example a truncated version of the transcript, there must be some additional formal evidence that this definition actually uniquely identifies sessions.

Definition 8 (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 $\Phi_i^s = \mathtt{accept}$ and $\Phi_j^t = \mathtt{accept}$ 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 and an adversary.

Security Game. Assume there is a global variable *pinfo* which stores the role information of each party for the considered protocol Π.[6] In the game, the following steps are performed:

1. Given the security parameter κ the challenger implements the collection of oracles $\{\pi_i^s : i, j \in [\ell], s \in [d]\}$ with respect to Π and *pinfo*. In this process, the challenger 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, the challenger gives the adversary $\mathcal{A}$ all identifiers $\{id_i\}$, all public keys (if any) and *pinfo* as input.

2. Next the adversary may start issuing $\mathsf{Send}^{\mathsf{pre}}$, RevealKey, Corrupt, Encrypt, Decrypt, and 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 providing 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 RegisterParty query and the extended Corrupt query.

Definition 9 (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[7] 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 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,

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

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

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

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

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

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 10 (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 9 and $\tau_i = \tau_j = \infty$.

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

Definition 11 (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 9 and $\tau_i, \tau_j > \tau_0$.

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. Server oracles may not be corrupted after accepting (such that for them security holds only in the basic sense of Definition 10). We will later show that TLS-RSA-PSK fulfills this extended definition of security.

Definition 12 (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 9 and it holds that $\tau_i = \infty$ and $\tau_j > \tau_0$ if π_i^s has internal state $\rho = \mathsf{Server}$ or $\tau_i > \tau_0$ 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 will analyse the security of each TLS-PSK ciphersuite with the following theorems respectively. Due to space restrictions, the complete proofs were postponed to the Appendix.

Theorem 1. *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 $(t, \epsilon_{\mathsf{PRF}})$-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 10 with $t \approx t'$ it holds that*

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

The full proof of this theorem is given in Appendix A.

Theorem 2. *Let μ be the output length of $\mathsf{PRF}_{\mathsf{TLS}}$ and let λ be the length of the nonces. Assume that the key derivation function $\mathsf{PRF}_{\mathsf{TLS}}$ 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 the TLS pre-master secret, and that the sLHAE scheme is $(t, \epsilon_{\mathsf{StE}})$-secure.*

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks `TLS_DHE_PSK` *protocol in the sense of Definition 11 with $t \approx t'$ holds that*

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

The proof of this theorem is given in Appendix B.

Theorem 3. *Let μ be the output length of $\mathsf{PRF}_{\mathsf{TLS}}$ and let λ be the length of the nonces. Assume that the key derivation function $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{DPRF}})$-secure DPRF when keyed with the pre-master secret pms with $pms := \mathsf{C}||\mathsf{V}||\mathsf{R}||len_{PSK}||\mathsf{PSK}$ (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 12 with $t \approx t'$ it holds that*

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

The proof of this theorem is given in Appendix C. As mentioned before, our result does not cover key transport with RSA-PKCS as used in TLS that is known to not provide IND-CCA security. We stress again that when proving that `TLS_RSA_PSK` is ACCE secure without forward secrecy, one does not require any security assumption on the PKE scheme. This is because security solely relies on the secrecy of the pre-shared secret. The proof is very similar to the proof of `TLS_PSK` and omitted therefore. Similarly, `TLS_DHE_PSK` can be proven ACCE secure without forward secrecy without relying on the DDH assumption.

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 most 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.[8] 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 value, the pre-shared secret or the

[8]The proof is probably closest to that of TLS-DH for mutual authentication (see KSS [28]) where both client and server have a static public key and the common secret is the Diffie-Hellman value of these keys. In some sense, one can think of this ciphersuite to be like TLS-PSK with pre-shared keys that are the Diffie-Hellman keys between the communication partners.

freshly exchanged secret, 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. New proofs for NMAC and HMAC. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619. Springer, 2006.

[3] 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.

[4] 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.

[5] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *IEEE Symposium on Security and Privacy*, pages 445–459. IEEE Computer Society, 2013.

[6] 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.

[7] 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.

[8] 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.

[9] U. Blumenthal and P. Goel. Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS). RFC 4785 (Proposed Standard), January 2007.

[10] Botan Software Developers. Botan, 2013. `http://botan.randombit.net/`.

[11] BouncyCastle Software Developers. Bouncy Castle Crypto APIs, 2013. `http://www.bouncycastle.org/`.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, Cascade and HMAC modes. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004.

[19] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005.

[20] Marc Fischlin, Anja Lehmann, and Daniel Wagner. Hash function combiners in TLS and SSL. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2010.

[21] 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`.

[22] Pierre-Alain Fouque, David Pointcheval, and Sébastien Zimmer. HMAC is a randomness extractor and applications to TLS. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08: 3rd Conference on Computer and Communications Security*, pages 21–32. ACM Press, March 2008.

[23] 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.

[24] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 387–398. ACM, 2013.

[25] 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.

[26] Ik Rae Jeong, Jonathan Katz, and Dong Hoon Lee. One-round protocols for two-party authenticated key exchange. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS 04: 2nd International Conference on Applied Cryptography and Network Security*, volume 3089 of *Lecture Notes in Computer Science*, pages 220–232. Springer, June 2004.

[27] 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.

[28] 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.

[29] 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.

[30] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997.

[31] 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.

[32] 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.

[33] 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.

[34] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, 2003.

[35] Nikos Mavrogiannopoulos and Simon Josefsson. The GnuTLS Transport Layer Security library. Last updated 2013-03-22, `http://gnutls.org`.

[36] Alfred Menezes, Minghua Qu, and Scott A. Vanstone. Some new key agreement protocols providing mutual implicit authentication. *SecondWorkshop on Selected Areas in Cryptography (SAC 95)*, 1995.

[37] Alfred Menezes and Nigel P. Smart. Security of signature schemes in a multi-user setting. *Des. Codes Cryptography*, 33(3):261–274, 2004.

[38] 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.

[39] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. The TLS handshake protocol: A modular analysis. *Journal of Cryptology*, 23(2):187–223, April 2010.

[40] National Institute of Standards and Technology. FIPS 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standard (FIPS), Publication 198. Technical report, 2002.

[41] OpenSSL. The OpenSSL project, 2013. `http://www.openssl.org`.

[42] L.Cogneau P. Urien and P. Martin. EMV support for TLS-PSK. draft-urien-tls-psk-emv-02, February 2011.

[43] 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.

[44] Paul Bakker. PolarSSL, 2013. `https://polarssl.org/`.

[45] PeerSec Networks. MatrixSSL, 2013. `http://www.matrixssl.org/`.

[46] Peter Gutmann. cryptlib, 2013. `https://www.cs.auckland.ac.nz/~pgut001/cryptlib/`.

[47] Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120, IBM, 1999.

[48] Todd Ouska. CyaSSL, 2013. `http://www.wolfssl.com/yaSSL/Home.html`.

[49] Pascal Urien. Introducing TLS-PSK authentication for EMV devices. In Waleed W. Smari and William K. McQuay, editors, *CTS*, pages 371–377. IEEE, 2010.

[50] Berkant Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Des. Codes Cryptography*, 46(3):329–342, 2008.

A Proof of Theorem 1

We first partition the set of all adversaries into two categories:

1. Adversaries that succeed in making an oracle accept maliciously. We call such an adversary an authentication-adversary.

2. Adversaries that do not succeed in making any oracle accept maliciously, but which answer the encryption-challenge. We call such an adversary an encryption-adversary.

We prove Theorem 1 by the following two lemmas. Lemma 3 bounds the probability ϵ_{auth} that an authentication-adversary succeeds, Lemma 4 bounds the probability ϵ_{enc} that an encryption-adversary succeeds. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}}$$

Lemma 3. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously is at most*

$$\epsilon_{\mathsf{auth}} \leq (d\ell)^2 \cdot \left(\frac{1}{2^\lambda} + 2\epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu} + 2\epsilon_{\mathsf{StE}} \right)$$

where all quantities are defined as stated in Theorem 1.

Note that ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously.

Proof. The proof proceeds in a *sequence of games*, where the first game is the real security experiment. We then describe several intermediate games that modify the original game step-by-step, and argue that our complexity assumptions imply that each game is computationally indistinguishable from the previous one. We end up in the final game, where no adversary can break the security of the protocol. Let S_δ^1 be the event that occurs when the first oracle that accepts maliciously in the sense of Definition 10 in Game δ.

Game 0. This game equals the ACCE security experiment described in Section 4. Thus, for some ϵ_{auth} we have

$$\Pr[\mathsf{S}_0^1] = \epsilon_{\mathsf{auth}}$$

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_i or r_j which is not unique.

$$\Pr[\mathsf{S}_0^1] \leq \Pr[\mathsf{S}_1^1] + \frac{(d\ell)^2}{2^\lambda}$$

Note that now each oracle has a unique nonce r_i or r_j, which is included in the signatures. We will use this to ensure that each oracle that accepts with non-corrupted partner has a *unique* partner oracle.

Game 2. We try to guess which oracle will be the first oracle to accept maliciously and its partner oracle. If our guess is incorrect, i.e. if there is another oracle that accepts before, then we abort this game.

Technically, this game is identical to previous game, except for the following modifications. The challenger guesses two random indices $(i^*, s^*, j^*, t^*) \xleftarrow{\$} [\ell]^2 \times [d]^2$. If there exists an oracle π_i^s that 'accepts' maliciously with intended communication partner P_j (i.e. oracle π_j^t), and $(i, s, j, t) \neq (i^*, s^*, j^*, t^*)$, then the challenger aborts the game. Note that if π_i^s is the first oracle that 'accepts' maliciously, then with probability $1/(d\ell)^2$ we have $(i, s, j, t) = (i^*, s^*, j^*, t^*)$, and thus

$$\Pr[\mathsf{S}_1^1] \leq (d\ell)^2 \cdot \Pr[\mathsf{S}_2^1]$$

Note that in this game the attacker can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first oracle that 'accepts' maliciously, as otherwise the game is aborted.

Game 3. In this game we replace the master secret ms computed by $\pi_{i^*}^{s^*}$ with an independent random value $\widetilde{ms}$. Moreover, if $\pi_{j^*}^{t^*}$ computes the master key using the same nonces $r_i^*||r_j^*$ as $\pi_{i^*}^{s^*}$, then we set its master key to $\widetilde{ms}$ as well. We make use of the fact that the pre-shared keys PSK are chosen uniformly at random from the key space of $\mathsf{PRF}_{\mathsf{TLS}}$. Distinguishing Game 3 from Game 2 implies an algorithm breaking the security of the pseudo-random function $\mathsf{PRF}_{\mathsf{TLS}}$, thus

$$\Pr[\mathsf{S}_2^1] \leq \Pr[\mathsf{S}_3^1] + \epsilon_{\mathsf{PRF}}$$

Game 4. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ with a random function RF. If $\pi_{j^*}^{t^*}$ uses the same master secret $\widetilde{ms}$ as $\pi_{i^*}^{s^*}$ (cf. Game 3), then the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{j^*}^{t^*}$ is replaced as well. In particular, this function is used to compute the `Finished` messages by both partner oracles.

Distinguishing Game 4 from Game 3 implies an algorithm breaking the security of the pseudo-random function $\mathsf{PRF}_{\mathsf{TLS}}$, thus

$$\Pr[\mathsf{S}_3^1] \leq \Pr[\mathsf{S}_4^1] + \epsilon_{\mathsf{PRF}}$$

Game 5. In Game 4 we have replaced the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ with a random function RF. Thus, the *Finished* message is $\mathsf{RF}(\widetilde{ms}, \mathit{label}_3||\mathsf{H}(m_1||\ldots))$, where $(m_1||\ldots)$ denotes the transcript of all messages sent and and received by oracle $\pi_{i^*}^{s^*}$. In this game, the challenger proceeds exactly like the challenger in Game 4, except that we add an abort rule. We abort the game, if oracle $\pi_{i^*}^{s^*}$ ever evaluates the random function RF on an input m^* such that $\mathsf{H}(m^*) = \mathsf{H}(m_1||\ldots)$, where $(m^* \neq m_1||\ldots)$. Since $\mathsf{H}(m^*) = \mathsf{H}(m_1||\ldots)$ implies that a collision for the hash function H is found, we have

$$\Pr[\mathsf{S}_4^1] \leq \Pr[\mathsf{S}_5^1] + \epsilon_{\mathsf{H}}$$

Game 6. Finally we use that the full transcript of all messages sent and received is used to compute the `Finished` messages, and that `Finished` messages are computed by evaluating a truly random function that is only accessible to $\pi_{i^*}^{s^*}$ and (possibly) $\pi_{j^*}^{t^*}$ due to Game 5. This allows to show that any adversary has probability at most $\frac{1}{2^\mu}$ of making oracle $\pi_{i^*}^{s^*}$ accept without having a matching conversation to $\pi_{j^*}^{t^*}$. The `Finished` messages are computed by evaluating a truly random function $\mathsf{RF}_{\widetilde{ms}}(\cdot)$, which is only accessible to oracles sharing $\widetilde{ms}$, and the full transcript containing all previous messages is used to compute the `Finished` messages. If there is no oracle having a matching conversation to $\pi_{i^*}^{s^*}$, the adversary receives no information about $\mathsf{RF}_{\widetilde{ms}}(\cdot)$. Therefore we have

$$\Pr[\mathsf{S}_5^1] \leq \Pr[\mathsf{S}_6^1] + \frac{1}{2^\mu}$$

Game 7. In this game we show that any successful adversary can be used to break the sLHAE-security of the encryption system. This step is necessary, as an adversary can violate the matching conversations (MC) definition (and thus make an oracle accept maliciously) by creating a new, valid encryption (C_C or C_S) of one of the `Finished` messages (fin_C or fin_S), which is distinct from the ciphertext output by the corresponding oracle (*Client* or *Server*) or of any other messages sent later. Therefore, we need to make sure that $\mathcal{A}$ is not able to generate new, valid symmetric encryptions

of the `Finished` messages. To this end we exploit the properties of the sLHAE scheme. The forged ciphertexts produced by the adversary are either computed using $k_{\mathsf{enc}}^{\mathsf{Client}}$ or using $k_{\mathsf{dec}}^{\mathsf{Client}}$. The challenger can guess in which of the two keys are used with probability at least 1/2. On failure, it simply aborts. On success, the challenger can embed the sLHAE challenge into this key.

$$\Pr[\mathsf{S}_6^1] \leq 2\Pr[\mathsf{S}_7^1]$$

According to the sLHAE security of the symmetric encryption scheme, $\mathcal{A}$ has advantage at most ϵ_{StE} in breaking the sLHAE security. The access to the oracles in the sLHAE security game can directly be used to implement the $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ query of the ACCE security game. Observe that the values generated in this game are exactly distributed as in the previous game. We have

$$\Pr[\mathsf{S}_7^1] = \epsilon_{\mathsf{StE}}.$$

Collecting probabilities from Game 0 to Game 7 yields Lemma 3. □

Lemma 4. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the encryption-challenge correctly in the sense of Definition 10 is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + (d\ell)^2 \left(2\epsilon_{\mathsf{PRF}} + 2\epsilon_{\mathsf{StE}}\right)$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 10 (cf. Lemma 3) and all other quantities are defined as stated in Theorem 1.

Proof. Assume without loss of generality that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 10 are satisfied. Let S_δ^2 denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{S}_\delta^2] - 1/2$ denote the advantage of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment described in Section 4. For some ϵ_{enc} we have

$$\Pr[\mathsf{S}_0^2] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = 1/2 + \mathsf{Adv}_0$$

Game 1. The challenger in this game proceeds as before, but it aborts if the test oracle accepts without a unique partner oracle. In other words, in this game, we make the same modifications as in Game 0 to Game 7 in the proof of Lemma 3. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}}$$

We note that at this this point we have now excluded active adversaries between and, moreover, for all τ and any τ-fresh oracle π_i^s there is a unique oracle π_j^t such that π_i^s and π_j^t have matching conversations. Therefore, any accepting oracle has an uniquely identified partner oracle.

Game 2. Technically, this game is identical to the previous game, except for the following modifications. The challenger aborts if it fails to guess the oracle $\pi_{i^*}^{s^*}$ (and its partner $\pi_{j^*}^{t^*}$) that the adversary attacks. The probability that the challenger guesses correctly is at least $1/(d\ell)^2$ we have

$$\mathsf{Adv}_1 \leq (d\ell)^2 \cdot \mathsf{Adv}_2$$

Game 3. In this game we replace the master secret ms computed by $\pi_{i^*}^{s^*}$ with an independent random value $\widetilde{ms}$. Moreover, if $\pi_{j^*}^{t^*}$ compute the master key using same nonces $r_i^*||r_j^*$ as $\pi_{i^*}^{s^*}$, then we set its master key as $\widetilde{ms}$. We make use of the fact that each pre-shared key is chosen uniformly at random from the key space of $\mathsf{PRF}_{\mathsf{TLS}}$. Distinguishing Game 3 from Game 2 implies an algorithm breaking the security of the pseudo-random function $\mathsf{PRF}_{\mathsf{TLS}}$, thus

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{PRF}}$$

Game 4. As in Game 4 in the proof of Lemma 4, we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ to compute the application keys with a random function $\mathsf{RF}_{\widetilde{ms}}$. In particular, this function is used to compute the key material as

$$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := \mathsf{RF}_{\widetilde{ms}}(label_2||r_i||r_j)$$

Distinguishing Game 4 from Game 3 again implies an algorithm breaking the security of the pseudo-random function $\mathsf{PRF}_{\mathsf{TLS}}$, thus we have

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{PRF}}$$

Game 5. Now we use that the key material $K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C}$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ in the stateful symmetric encryption scheme is uniformly random and independent of all TLS Handshake messages exchanged in the pre-accept phase. The challenger can again guess (with probability at least 1/2) the key that is used to create the ciphertexts which the adversary attacks (either $k_{\mathsf{enc}}^{\mathsf{Client}}$ or $k_{\mathsf{dec}}^{\mathsf{Client}}$) and embedd the sLHAE challenge in it.

So we construct a simulator $\mathcal{B}$ that uses a successful ACCE attacker $\mathcal{A}$ to break the security of the underlying sLHAE secure symmetric encryption scheme (see Section 2.4). By assumption, the simulator $\mathcal{B}$ is given access to an encryption oracle $\mathsf{Encrypt}$ and a decryption oracle $\mathsf{Decrypt}$. $\mathcal{B}$ embeds the sLHAE experiment by simply forwarding all $\mathsf{Encrypt}(\pi_{i^*}^{s^*}, \cdot)$ queries to $\mathsf{Encrypt}$, and all $\mathsf{Decrypt}(\pi_{j^*}^{t^*}, \cdot)$ queries to $\mathsf{Decrypt}$. Otherwise it proceeds as the challenger in Game 4.

As before we have

$$\mathsf{Adv}_4 = 2 \cdot \mathsf{Adv}_5.$$

If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then $\mathcal{B}$ forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the simulator essentially relays all messages it is easy to see that an attacker $\mathcal{A}$ having advantage ϵ' yields an attacker $\mathcal{B}$ against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$. Since by assumption any attacker has advantage at most ϵ_{StE} in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_5 \leq \epsilon_{\mathsf{StE}}$$

□

Summing up probabilities from Lemmas 3 and 4, we obtain that

$$\begin{aligned}\epsilon_{\mathsf{tls}} &\leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}} \\ &= (d\ell)^2 \left(\frac{1}{2^{\lambda-1}} + 6 \cdot \epsilon_{\mathsf{PRF}} + 2 \cdot \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu-1}} + 6 \cdot \epsilon_{\mathsf{StE}} \right),\end{aligned}$$

which proves Theorem 1.

B Proof of Theorem 2

Similar to the previous proof, we prove Theorem 2 via two lemmas. Lemma 5 bounds the probability ϵ_{auth} that an authentication-adversary succeeds, Lemma 6 bounds the probability ϵ_{enc} that an encryption-adversary succeeds. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}}$$

Lemma 5. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously in the sense of Definition 11 is at most*

$$\epsilon_{\mathsf{auth}} \leq (d\ell)^2 \cdot \left(\frac{1}{2^{\lambda}} + \epsilon_{\mathsf{DPRF}} + \epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + 2\epsilon_{\mathsf{StE}} \right)$$

where all quantities are defined as stated in Theorem 2.

Proof. The proof proceeds in a *sequence of games.*

Game 0. This game equals the AKE security experiment described in Section 4. Thus, for some ϵ_{auth} we have

$$\Pr[\mathsf{S}_0^3] = \epsilon_{\mathsf{auth}}.$$

Game 1. This game corresponds to Game 1 in the proof of Lemma 3. With the same arguments we have

$$\Pr[\mathsf{S}_0^3 \leq \Pr[\mathsf{S}_1^3] + \frac{(d\ell)^2}{2^{\lambda}}.$$

Game 2. This game corresponds to Game 2 in the proof of Lemma 3. We have that

$$\Pr[\mathsf{S}_1^3] \leq (d\ell)^2 \cdot \Pr[\mathsf{S}_2^3].$$

Game 3. In this game, we replace the master secret ms that is generated by $\pi_{i^*}^{s^*}$ with a with the output $\widetilde{ms}$ of a truly random function $\mathsf{RF}_{\mathsf{PSK},Z}$. Moreover, if $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ have computed the same random nonces and the same Diffie-Hellman value Z, we set the master secret of $\pi_{j^*}^{t^*}$ to $\widetilde{ms}$ as well. Otherwise we compute the master secret of $\pi_{j^*}^{t^*}$ as specified in the protocol. We exploit that $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{DPRF}})$-secure DPRF by showing that any adversary that recognizes our modification can be used to build a successful attacker against the DPRF properties of $\mathsf{PRF}_{\mathsf{TLS}}$. In the DPRF security game the simulator first calls $\mathsf{Init}(1)$, indicating that it wants to specify the Diffie-Hellman value when making its DPRF queries. Furthermore it will use the queries granted in the DPRF security game to compute the outputs of $\mathsf{PRF}_{\mathsf{TLS}}$. Now, if $\hat{b} = 0$ in the DPRF security game, we are in the previous game of the proof. In case $\hat{b} = 1$ we are in the current game of the proof. So, any adversary $\mathcal{A}$ that distinguishes Game 3 from Game 2 implies a DPRF-adversary $\mathcal{B}$ that breaks the DPRF-security of $\mathsf{PRF}_{\mathsf{TLS}}$.

We get that

$$\Pr[\mathsf{S}_2^3] \leq \Pr[\mathsf{S}_3^3] + \epsilon_{\mathsf{DPRF}}$$

Game 4. This game corresponds to Game 4 in the proof of Lemma 3. With the same arguments we have that

$$\Pr[\mathsf{S}_3^3] \leq \Pr[\mathsf{S}_4^3] + \epsilon_{\mathsf{PRF}}$$

Game 5. This game corresponds to Game 5 in the proof of Lemma 3. With the same arguments we have that

$$\Pr[\mathsf{S}_4^1] \leq \Pr[\mathsf{S}_5^1] + \epsilon_{\mathsf{H}}$$

Game 6. This game proceeds as Game 6 in the proof of Lemma 3. With same arguments as in Game 6 of Lemma 3, we have that the adversary cannot successfully compute finished messages except for some negligible probability. We have that

$$\Pr[\mathsf{S}_5^3] \leq \Pr[\mathsf{S}_6^3] + \frac{1}{2^\mu}.$$

Game 7. In this game we show that any successful adversary can be used to break the sLHAE-security of the encryption system. This game proceeds as the Game 7 in the proof of Lemma 3. With the same arguments as in Game 7 of Lemma 3, we have that

$$\Pr[\mathsf{S}_6^1] \leq 2 \cdot \Pr[\mathsf{S}_7^1]$$

and

$$\Pr[\mathsf{S}_7^1] \leq \epsilon_{\mathsf{StE}}.$$

Collecting probabilities from Game 0 to Game 7 yields Lemma 5. □

Lemma 6. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the encryption-challenge correctly in the sense of Definition 11 is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + (d\ell)^2 \left(\epsilon_{\mathsf{DDH}} + \epsilon_{\mathsf{DPRF}} + \epsilon_{\mathsf{PRF}} + 2\epsilon_{\mathsf{StE}}\right)$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 11 (cf. Lemma 5) and all other quantities are defined as stated in Theorem 2.

Proof. Assume without loss of generality that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 11 are satisfied. Let S_δ^4 denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{S}_\delta^4] - 1/2$ denote the advantage of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment described in Section 4. For some ϵ_{enc} we have

$$\Pr[\mathsf{S}_0^4] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = 1/2 + \mathsf{Adv}_0$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random, if there exists any oracle that accepts maliciously in the sense of Definition 11. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}}$$

where ϵ_{auth} an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 11 (cf. Lemma 5).

Recall that we assume that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 11 are satisfied. In particular it outputs (i, s, b') such that π_i^s 'accepts' after the τ_0-th query of $\mathcal{A}$ with intended partner P_j, and P_j is τ_j-corrupted with $\tau_j > \tau_0$. Note that in Game 1 for any such oracle π_i^s there exists a unique 'partner oracle' π_j^t such that π_i^s has a matching conversation to π_j^t, as the game is aborted otherwise.

Game 2. The challenger in this game proceeds as before, but in addition guesses the indices $(i^*, s^*, j^*, t^*) \xleftarrow{\$} [\ell]^2 \times [d]^2$ of the oracle $\pi_{i^*}^{s^*}$ for which the adversary will correctly answer the encryption challenge and its corresponding partner oracle $\pi_{j^*}^{t^*}$. It aborts on failure and proceeds otherwise. Thus,

$$\mathsf{Adv}_1 \leq (d\ell)^2 \cdot \mathsf{Adv}_2.$$

Game 3. Let $T_{i^*}^{s^*} = g^u$ denote the Diffie-Hellman share chosen by $\pi_{i^*}^{s^*}$, and let $T_{j^*}^{t^*} = g^v$ denote the share chosen by its partner $\pi_{j^*}^{t^*}$. Thus, both oracles compute the pre-master secret as $pms = len_Z \,||g^{uv}||\, len_{\mathsf{PSK}}||\, \mathsf{PSK}_{i^*,j^*}$. The challenger in this game proceeds as before, but replaces the DH key $Z = g^{uv}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*,i^*}^{t^*}$ with a random group element g^w for $w \xleftarrow{\$} \mathbb{Z}_q$. We then have that

$\widetilde{pms} = len_Z||g^w||len_{\mathsf{PSK}}||\mathsf{PSK}_{i^*,j^*}$. Recall that by assumption the DH key will at no time be revealed by the adversary (in contrast to the pre-shared keys).[9]

Suppose that there exists an algorithm $\mathcal{A}$ distinguishing Game 3 from Game 2. Then we can construct an algorithm $\mathcal{B}$ solving the DDH problem as follows. $\mathcal{B}$ receives as input (g, g^u, g^v, g^z). It implements the challenger for $\mathcal{A}$ as in Game 2, except that it sets $T_{i^*,s^*} := g^u$ and $T_{j^*,t^*} := g^v$, and the pre-master secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ equal to $pms := len_Z||g^z||len_{\mathsf{PSK}}||\mathsf{PSK}_{i^*,j^*}$. Note that $\mathcal{B}$ can simulate all messages exchanged between $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ properly, in particular the finished messages using knowledge of $pms = len_Z||g^z||len_{\mathsf{PSK}}||\mathsf{PSK}_{i^*,j^*}$. Since all other oracles are not modified, $\mathcal{B}$ can simulate these oracles properly as well. If $z = uv$, then the view of $\mathcal{A}$ when interacting with $\mathcal{B}$ is identical to Game 2, while if $z \xleftarrow{\$} \mathbb{Z}_q$ then it is identical to Game 3. Thus, the DDH assumption implies that

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{DDH}}$$

Game 4. Recall that $\pi_{i^*}^{s^*}$ computes the master secret as $ms = \mathsf{PRF}_{\mathsf{TLS}}(len_Z||Z||len_{\mathsf{PSK}}||\mathsf{PSK}_{i^*,j^*}, label_1||r_{i^*}||r_{j^*})$, where Z denotes the random group element chosen by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In this game we replace the master secret ms computed by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with an independent random value $\widetilde{ms}$. By assumption we have that $\mathsf{PRF}_{\mathsf{TLS}}$ constitutes a secure DPRF when at least one of the values Z or PSK is random and not revealed. Due to the security of $\mathsf{PRF}_{\mathsf{TLS}}$, we have that

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{DPRF}}$$

Game 5. We now replace the function $\mathsf{PRF}_{\mathsf{TLS}}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ to derive the application keys with a random function $\mathsf{RF}_{\widetilde{ms}}$. Of course the same random function is used for both oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In particular, this function is used to compute the key material as

$$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := \mathsf{RF}_{\widetilde{ms}}(label_2||r_i^*||r_j^*)$$

Distinguishing Game 5 from Game 4 again implies an algorithm breaking the security of the pseudo-random function $\mathsf{PRF}_{\mathsf{TLS}}$, thus we have

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + \epsilon_{\mathsf{PRF}}$$

Note that in Game 5 the key material $K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C}$ of oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is uniformly random and independent of all TLS Handshake messages exchanged in the pre-accept phase.

Game 6. Now we use that the key material $K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C}$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ in the stateful symmetric encryption scheme is uniformly random and independent of all TLS Handshake messages.

In this game, we construct a simulator $\mathcal{B}$ that uses a successful ACCE attacker $\mathcal{A}$ to break the security of the underlying sLHAE secure symmetric encryption scheme. By assumption, the simulator $\mathcal{B}$ is given access to an encryption oracle $\mathsf{Encrypt}$ and a decryption oracle $\mathsf{Decrypt}$. First,

[9] In the previous proof we excluded active adversaries. The result shows that as long as the pre-shared keys remain uncorrupted before the oracles accept, authentication can be guaranteed. In this proof we show that even if the pre-shared keys are revealed after the oracles have accepted, no adversary can break the PFS encryption-challenge.

the challenger chooses in which key ($k_{\mathsf{enc}}^{\mathsf{Client}}$ or $k_{\mathsf{dec}}^{\mathsf{Client}}$) it needs to embed the sLHAE challenge. With probability at least 1/2 its guess is correct. On failure it aborts. Again, we get

$$\mathsf{Adv}_5 = 2 \cdot \mathsf{Adv}_6$$

$\mathcal{B}$ simulates the ACCE experiment by simply forwarding all $\mathsf{Encrypt}(\pi_{i^*}^{s^*}, \cdot)$ queries to $\mathsf{Encrypt}$ in the sLHAE game, and all $\mathsf{Decrypt}(\pi_{j^*}^{t^*}, \cdot)$ queries to $\mathsf{Decrypt}$ in the sLHAE game. Otherwise it proceeds as the challenger in Game 5. Observe that the values generated in this game are exactly distributed as in the previous game.

If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then $\mathcal{B}$ forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the simulator essentially relays all messages it is easy to see that an attacker $\mathcal{A}$ having advantage ϵ' yields an attacker $\mathcal{B}$ against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$.

Since by assumption, any attacker has advantage at most ϵ_{StE} in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_6 \leq \epsilon_{\mathsf{StE}}$$

□

Summing up probabilities from Lemmas 5 and 6 proves Theorem 2.

C Proof of Theorem 3

In the proof of this theorem, we consider the following two lemmas. Lemma 7 bounds the probability ϵ_{auth} that an adversary makes an oracle accept maliciously. Lemma 8 bounds the probability ϵ_{enc} that an adversary breaks the APFS encryption challenge. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{enc}} + \epsilon_{\mathsf{auth}}$$

We prove Theorem 3 via the following lemmas.

Lemma 7. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously in the sense of Definition 12 is at most*

$$\epsilon_{\mathsf{auth}} \leq (d\ell)^2 \cdot \left(\frac{1}{2^\lambda} + \epsilon_{\mathsf{DPRF}} + \epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu} + 2\epsilon_{\mathsf{StE}} \right)$$

where all quantities are defined as stated in Theorem 1.

The bound on ϵ_{auth} is derived almost exactly like in the proof of Lemma 5, therefore we omit the details.

Lemma 8. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the encryption-challenge correctly in the sense of Definition 12 is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + (d\ell)^2 \left(\epsilon_{\mathsf{PKE}} + \epsilon_{\mathsf{DPRF}} + \epsilon_{\mathsf{PRF}} + 2\epsilon_{\mathsf{StE}} \right)$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 12 (cf. Lemma 7) and all other quantities are defined as stated in Theorem 3.

PROOF. Assume without loss of generality that $\mathcal{A}$ always outputs (i, j, s, b') such that all conditions in Property 2 of Definition 12 are satisfied. Let S_δ^6 denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{S}_\delta^6] - 1/2$ denote the advantage of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment. For some ϵ_{enc} we have

$$\Pr[\mathsf{S}_0^6] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = 1/2 + \mathsf{Adv}_0$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random, if there exists any oracle that accepts maliciously in the sense of Definition 12. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}}$$

where ϵ_{auth} an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 12 (cf. Lemma 7).

Recall that we assume that $\mathcal{A}$ always outputs (i, j, s, b') such that all conditions in Property 2 of Definition 12 are satisfied. Note that in Game 1 for any such oracle π_i^s there exists a unique 'partner oracle' π_j^t such that π_i^s has a matching conversation to π_j^t, as the game is aborted otherwise.

Game 2. The challenger in this game proceeds as before, but in addition guesses the oracle $\pi_{i^*}^{s^*}$ (and its partner oracle $\pi_{j^*}^{t^*}$) for which the adversary breaks the APFS encryption challenge by drawing random the indices $(i, s, j, t) \xleftarrow{\$} [\ell]^2 \times [d]^2$. With probability $1/(d\ell)^2$ we have $(i, s, j, t) = (i^*, s^*, j^*, t^*)$, and thus

$$\mathsf{Adv}_1 \leq (d\ell)^2 \cdot \mathsf{Adv}_2$$

Game 3. In this game we replace the ciphertext c^* sent by the client oracle $\pi_{i^*}^{s^*}$, by a random ciphertext c' of a truly random message. However, the oracle $\pi_{i^*}^{s^*}$ and its partner oracle (if it exists) use a random nonce R^* which is independent of c' to compute the master key. More precisely, since the challenger implements all server oracles it can, whenever the ciphertext c' is received by any server oracle of P_{j^*}, make it use R^*. We show that this modification is indistinguishable from the previous game when the PKE is secure. Any adversary that can distinguish these two games can be used to break the security of the public key scheme as follows. We now embed the challenge public key of the PKE challenger in pk_{j^*}. For all other oracles $\pi_{j^*}^t$ of P_{j^*} with $t \in [d]$ and $t \neq t^*$ we use the decryption queries granted by the PKE challenger to decrypt c messages. We sent R^* after it is drawn by the client to the PKE challenger who returns a ciphertext c'. Next we send c' to the server oracle $\pi_{j^*}^{t^*}$. Observe that if c' is an encryption of R^* we are in the previous game. However, if c' encrypts an independently drawn random message we are in the current game. So any attacker that distinguishes these two games can directly be used to break the security of the PKE scheme.

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{PKE}}$$

Game 4. Recall that $\pi_{i*}^{s^*}$ computes the master secret as $ms = \mathsf{PRF}_{\mathsf{TLS}}(\mathsf{C}||\mathsf{V}||\mathsf{R}^*||len_{\mathsf{PSK}}||\mathsf{PSK}_{i^*}, label_1||r_{i^*}||r_{j^*})$. In this game we replace the master secret ms computed by $\pi_{i*}^{s^*}$ with an independent random value $\widetilde{ms}$. Moreover, as $\pi_{j*}^{t^*}$ receives as input the same ciphtertext c' that was sent from $\pi_{i*}^{s^*}$, we set the master secret of $\pi_{j*}^{t^*}$ equal to $\widetilde{ms}$ as well. We exploit that by assumption $\mathsf{PRF}_{\mathsf{TLS}}$ is a $(t, \epsilon_{\mathsf{DPRF}})$-secure DPRF. Therefore, as long as at least one of the values R^* and PSK_{i^*} are chosen at random and are not revealed, the master secret is indistinguishable from random. Any adversary that recognizes our modification can be used to break the security of $\mathsf{PRF}_{\mathsf{TLS}}$.

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{DPRF}}$$

Game 5. This game proceeds as the same as the Game 5 in the proof of Lemma 6. With the same arguments, we have that

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + \epsilon_{\mathsf{PRF}}$$

Game 6. This game proceeds as the same as the Game 6 in the proof of Lemma 6. With the same arguments, we have that

$$\mathsf{Adv}_5 \leq 2\mathsf{Adv}_6$$

and

$$\mathsf{Adv}_6 = \epsilon_{\mathsf{StE}}.$$

□

Summing up probabilities from Lemmas 7 and 8 proves Theorem 3.