On the Security of TLS-DHE in the Standard Model

Tibor Jager[1], Florian Kohlar[2], Sven Schäge[3,⋆], and Jörg Schwenk[2]

[1] Karlsruhe Institute of Technology, Germany
tibor.jager@kit.edu
[2] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
{florian.kohlar,joerg.schwenk}@rub.de
[3] University College London, United Kingdom
s.schage@ucl.ac.uk

Abstract. TLS is the most important cryptographic protocol in use today. However, up to now there is no complete cryptographic security proof in the standard model, nor in any other model. We give the first such proof for the core cryptographic protocol of TLS ciphersuites based on ephemeral Diffie-Hellman key exchange (TLS-DHE), which include the cipher suite `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA` mandatory in TLS 1.0 and TLS 1.1. It is impossible to prove security of the TLS Handshake protocol in any classical key-indistinguishability-based security model (like for instance the Bellare-Rogaway or the Canetti-Krawczyk model), due to subtle issues with the encryption of the final `Finished` messages. Therefore we start with proving the security of a truncated version of the TLS-DHE Handshake protocol, which has been considered in previous works on TLS. Then we define the notion of authenticated and confidential channel establishment (ACCE) as a new security model which captures precisely the security properties expected from TLS in practice, and show that the combination of the TLS Handshake with data encryption in the TLS Record Layer can be proven secure in this model.

Keywords: authenticated key exchange, SSL, TLS, provable security, ephemeral Diffie-Hellman.

1 Introduction

Transport Layer Security (TLS) is the single most important Internet security mechanism today. Session keys in TLS are established in the TLS Handshake protocol, using either encrypted key transport (TLS-RSA) or (ephemeral) Diffie-Hellman key exchange (TLS-DH(E)), whereas authentication can be provided mutual or server-only. Due to a subtle interleaving of the TLS Handshake with the TLS Record Layer it is impossible to prove the security of TLS using well-established security models [4,10,9], which define security via indistinguishability of keys (see [18] for a detailed description of this issue). Therefore there is no security proof for the complete protocol up to now.

⋆ Supported by EPSRC grant number EP/G013829/1.

The paradox that the most important authenticated key-exchange (AKE) protocol cannot be proven secure in any existing security model can be solved in two ways. Either one considers a modified version of the TLS Handshake protocol ('truncated TLS'), which was subject to previous work [20], or a new security model for the combination of TLS Handshake protocol and data encryption in the TLS Record Layer must be devised. In this paper we follow both approaches.

1.1 Contributions

We provide new security results for the core cryptographic protocol of TLS based on ephemeral Diffie-Hellman key exchange (TLS-DHE).

First we give a formal proof that the truncated version of the TLS-DHE Handshake protocol from [20] is a secure authenticated key exchange protocol. We consider a security model which extends the well-known Bellare-Rogaway model [4] to adaptive corruptions and perfect forward secrecy in the public-key setting (cf. [6]). This allows to compare our results to previous work.

Second we define the notion of authenticated and confidential channel establishment (ACCE). ACCE protocols are an extension of AKE protocols, in the sense that the symmetric cipher is integrated into the model. In contrast to AKE protocols, where one requires *key indistinguishability*, we demand that a secure ACCE protocol allows to establish a 'secure communication channel' in the sense of stateful length-hiding authenticated encryption [22]. Loosely speaking, an ACCE channel guarantees that messages written to this channel are confidential (indistinguishable, and even the length of messages is concealed up to some granularity), and that a sequence of messages read from this channel corresponds exactly to the sequence of messages sent by the legitimate sender (of course up to dropping messages at the very end of the sequence, which is always possible). This captures exactly the properties expected from TLS-like protocols in practice. We prove that the combination of the TLS Handshake protocol with the TLS Record Layer forms a secure ACCE protocol, if the TLS Record Layer provides security in the sense of length-hiding authenticated encryption. Note that the latter was proven recently by Paterson *et al.* [22] for CBC-based Record Layer protocols.

The analyses of both truncated TLS-DHE (as an AKE protocol) and TLS-DHE (as an ACCE protocol) require, that the building blocks of TLS-DHE (digital signature scheme, Diffie-Hellman key exchange, symmetric cipher) meet certain security properties. The majority of these properties are standard assumptions, solely for the pseudo-random function we require an additional non-standard security assumption, which is a variant of the Oracle Diffie-Hellman assumption introduced by Abdalla, Bellare, and Rogaway [1]. We explain in Section 6 why such an assumptions seems hard to avoid. Our proof is stated for *mutual authentication*, i.e., the client authenticates itself using a client certificate. This allows us to base our work on standard definitions for secure authenticated key exchange.

1.2 Interpretation

Our results show that the core cryptographic protocol of TLS-DHE is cryptographically sound, if its building blocks are suitably secure (the full version [18] of this paper contains an analysis to what extent the concrete building blocks of TLS meet the required properties, here we can build upon previous work that considered particular components of TLS).

We note that TLS-DHE is much less used in practice than TLS with encrypted key transport (TLS-RSA). Moreover, we consider mutual authentication (that is, both the client and the server are in possession of a certified public key, which is used in the protocol to mutually authenticate each other), which is also rarely used in practice. We believe that our analysis of TLS-DHE is nevertheless of practical value, for the following reasons:

First, the TLS-DHE-based ciphersuite `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA` is mandatory for TLS 1.0 and 1.1, which are both still in widespread use. Only the most recent version TLS 1.2 prescribes TLS-RSA as mandatory. So one could theoretically configure a considerable amount of servers to use only TLS-DHE and benefit from the provable security guarantees of TLS-DHE as provided in our security analysis.

Second, we can show that TLS-DHE provides *perfect forward secrecy* – a very strong form of security, which basically states that future compromises of long-term secrets do no threaten past communication sessions. With encrypted key transport, as in TLS-RSA, this is not achievable, since an attacker that compromises the long-term key (the private decryption key) can easily obtain session keys from previous sessions by just decrypting recorded ciphertexts. To better protect users from the consequences of such key compromise attacks and offer better long-term security, service providers might therefore consider to switch to the (exclusive) use of TLS-DHE. Recently, Google has made a first step in that direction, by announcing that it will switch over to TLS-DHE as the default key exchange method for its services to provide (and push) perfect forward secrecy [2].

Third, it seems that giving a security proof of the actually most widespread option TLS-RSA is impossible in the standard model. Any approach we can think of would require IND-CCA-security of the encryption scheme used to transport the premaster secret from the client to the server, as otherwise we cannot simulate protocol executions while still being able to argue with indistinguishability of premaster secrets. But unfortunately it is well-known that the RSA-PKCS v1.5 scheme used in TLS is vulnerable to chosen-ciphertext attacks [7]. This problem was circumvented in previous work by either using an abstract public-key encryption scheme which is IND-CCA-secure [20], or by assuming PKCS#1 v2.0 (RSA-OAEP), which is not used in TLS, and omitting authentication [17].

Our work can also be seen as a 'stepping stone' towards a TLS version with a complete security proof in the standard model. Essentially, we identify certain security properties and prove that the TLS protocol framework yields a secure ACCE protocol under the assumption that the TLS building blocks satisfy these properties.

1.3 Related Work

Because of its eminent role, TLS and its building blocks have been subject to several security analyses. We mention only the works closely related to ours here, a more complete overview can be found in [18].

Gajek *et al.* [17] presented the first security analysis of the complete TLS protocol, combining Handshake and Record Layer, in the UC framework [9] for all three key exchange protocols static Diffie-Hellman, ephemeral signed Diffie-Hellman, and encrypted key transport. The ideal functionalities described in this paper are much weaker than the security guarantees we expect from TLS, since only unauthenticated key exchange is considered. The paper furthermore assumes that RSA-OAEP is used for encrypted key transport, which is not the case for current versions of TLS.

Morissey *et al.* [20] analysed, in a paper that is closest to our results, the security of the truncated TLS Handshake protocol in the random oracle model and provided a modular proof of security. They make extensive use of the random oracle model to separate the three layers in the TLS Handshake they define, and to switch from computational to indistinguishability based security models. The use of the random oracle model is justified by the authors of [20] since it seems impossible to prove the PKCS#1 v1.5 based ciphersuites of TLS secure in the standard model. This argumentation does not affect our work, since we consider Diffie-Hellman-based ciphersuites.

Paterson *et al.* [22] introduce the notion of length-hiding authenticated encryption, which captures the properties expected from the data encryption in the TLS Record Layer. Most importantly, they were able to show that CBC-based ciphersuites of TLS 1.1 and 1.2 meet this security notion. This work matches nicely our results on the TLS Handshake protocol, and is an important building block for our work.

Very recently, Brzuska *et al.* [8] proposed relaxed game-based security notions for key exchange. This approach may serve as an alternative to our ACCE-based approach to circumvent the impossibility of proving the TLS Handshake protocol secure in a key-indistinguishability-based security model.

1.4 Remark on Our Choice of the Security Model

Authenticated key exchange (AKE) is a basic building block in modern cryptography. However, since many different security models for different purposes exist [3,4,6,9,10,13,19,12], the choice of the right model is not an easy task, and must be considered carefully. We have to take into account that we cannot modify any detail in the TLS protocol, nor in the network protocols preceding it. We have chosen an enhanced variant of the first model of Bellare and Rogaway [4]. Variants of this model have also been studied by [12,6], and especially by [20]. Detailed reasons for our choice are given in the full version [18].

2 Preliminaries and Definitions

We denote with $\emptyset$ the empty string, and with $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ the set of integers between 1 and n. If A is a set, then $a \overset{\$}{\leftarrow} A$ denotes the action of sampling a uniformly random element from A. If A is a probabilistic algorithm, then $a \overset{\$}{\leftarrow} A$ denotes that A is run with fresh random coins and returns a. In addition to the complexity assumption described in the sequel, we need the standard security notions of digital signatures (EUF-CMA), pseudo-random functions, and the Decisional Diffie-Hellman (DDH) assumption. These are detailed in the full version [18].

The PRF-Oracle-Diffie-Hellman ($\mathsf{PRF\text{-}ODH}$*) Assumption.* Let G be a group with generator g. Let PRF be a deterministic function $z = \mathsf{PRF}(X, m)$, taking as input a key $X \in G$ and some bit string m, and returning a string $z \in \{0,1\}^{\mu}$. Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The adversary $\mathcal{A}$ outputs a value m.
2. The Challenger samples $u, v \xleftarrow{\$} [q]$, $z_1 \xleftarrow{\$} \{0,1\}^{\mu}$ uniformly random and sets $z_0 := \mathsf{PRF}(g^{uv}, m)$. Then it tosses a coin $b \in \{0,1\}$ and returns z_b, g^u and g^v to the adversary.
3. The adversary may query a pair (X, m') with $X \neq g^u$ to the challenger. The challenger replies with $\mathsf{PRF}(X^v, m')$.
4. Finally the adversary outputs a guess $b' \in \{0,1\}$.

Definition 1. *We say that the* $\mathsf{PRF\text{-}ODH}$ *problem is* $(t, \epsilon_{\mathsf{prfodh}})$-hard *with respect to* G *and* PRF*, if for all adversaries* $\mathcal{A}$ *that run in time* t *it holds that*

$$|\Pr[b = b'] - 1/2| \leq \epsilon_{\mathsf{prfodh}}.$$

The PRF-Oracle-Diffie-Hellman ($\mathsf{PRF\text{-}ODH}$) assumption is a variant of the ODH assumption introduced by Abdalla, Bellare and Rogaway in [1], adopted from hash functions to PRFs. In contrast to allowing a polynomial number of queries as in the original assumption [1], we allow only a single oracle query.

Stateful Length-Hiding Authenticated Encryption. The following description and security model was obtained from the authors of [22] via personal communication. See [22] for a detailed discussion and motivation of this security notion.

A *stateful symmetric encryption scheme* consists of two algorithms $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$. Algorithm $(C, st_e') \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m, st_e)$ takes as input a secret key $k \in \{0,1\}^{\kappa}$, an output ciphertext length $\mathsf{len} \in \mathbb{N}$, some header data $H \in \{0,1\}^*$, a plaintext $m \in \{0,1\}^*$, 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. Algorithm $(m', st_d') = \mathsf{StE.Dec}(k, H, C, st_d)$ 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.

Definition 2. *We say that a stateful symmetric encryption scheme* $\mathsf{StE} = (\mathsf{StE.Init}, \mathsf{StE.Enc}, \mathsf{StE.Dec})$ *is* $(t, \epsilon_{\mathsf{sLHAE}})$*-secure, if* $\Pr[b = b'] \leq \epsilon_{\mathsf{sLHAE}}$ *for all adversaries* $\mathcal{A}$ *running in time at most* t *in the following experiment.*

```
$\mathsf{Encrypt}(m_0, m_1, \mathsf{len}, H)$:
$u := u + 1$
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_0, st_e)$
$(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_1, st_e)$
If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$
$(C_u, st_e) := (C^{(b)}, st_e^{(b)})$
Return $C_u$
```

```
$\mathsf{Decrypt}(C, H)$:
$v := v + 1$
If $b = 0$, then return $\perp$
$(m, st_d) = \mathsf{StE.Dec}(k, H, C, st_d)$
If $v > u$ or $C \neq C_v$, then $\mathsf{phase} := 1$
If $\mathsf{phase} = 1$ then return $m$
Return $\perp$
```

Fig. 1. Encrypt and Decrypt oracles in the stateful LHAE security experiment

- *Choose* $b \xleftarrow{\$} \{0,1\}$ *and* $k \xleftarrow{\$} \{0,1\}^\kappa$, *and set* $st_e := \emptyset$ *and* $st_d := \emptyset$,
- *run* $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Encrypt},\mathsf{Decrypt}}$.

Here $\mathcal{A}^{\mathsf{Encrypt},\mathsf{Decrypt}}$ *denotes that* $\mathcal{A}$ *has access to two oracles* Encrypt *and* Decrypt. *The encryption oracle* $\mathsf{Encrypt}(m_0, m_1, \mathsf{len}, H)$ *takes as input two messages* m_0 *and* m_1, *length-parameter* len *and header data* H. *It maintains a counter* u *which is initialized to* 0. *Oracle* $\mathsf{Decrypt}(C, H)$ *takes as input a ciphertext* C *and header* H, *and keeps a counter* v *and a variable* phase, *both are initialized to* 0. *Both oracles process a query as defined in Figure 1.*

3 Transport Layer Security

The current version of TLS is 1.2 [16] coexists with its predecessors TLS 1.0 [14] and TLS 1.1 [15]. In the following we give a description of all messages sent during the TLS Handshake with ephemeral Diffie-Hellman key exchange and client authentication (i.e. for ciphersuites `TLS_DHE_*`). This description and its illustration in Figure 2 are valid for *all* TLS versions since v1.0. Our description makes use of several 'state variables' $(\Lambda, k, \Pi, \rho, st)$. For instance, variable $\Lambda \in \{\texttt{accept}, \texttt{reject}\}$ determines whether one party 'accepts' or 'rejects' an execution of the protocol, or variable k stores the session key. These variables will also appear later in our security model (Section 4).

The TLS Handshake protocol consists of 13 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. All messages are sent through the TLS Record Layer, which at startup provides no encryption nor any other cryptographic transformations.

Message m_1 is the `Client Hello` message. It contains four values, two of which are optional. 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. 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.

$\boxed{\mathsf{C}}$ $(I_C = pk_C, sk_C)$ $\boxed{\mathsf{S}}$ $(I_S = pk_S, sk_S)$

$r_C \xleftarrow{\$} \{0,1\}^\lambda$

$$\xrightarrow{m_1 := (r_C, \texttt{cs-list})}$$

$r_S \xleftarrow{\$} \{0,1\}^\lambda, t_S \xleftarrow{\$} \mathbb{Z}_q, T_S := g^{t_S} \mod p$

$\sigma_S := \mathsf{SIG.Sign}(sk_S, r_C || r_S || p || g || T_S)$

$$\xleftarrow{(m_2, m_3) := (r_S, \texttt{cs-choice}, cert_S)}$$

$$\xleftarrow{(m_4, m_5, m_6) := (p, g, T_S, \sigma_S, \texttt{get-cert}, done)}$$

$\Pi := S$, S is determined from $cert_S$

If $\mathsf{SIG.Vfy}(pk_\Pi, \sigma_S, r_C || r_S || p || g || T_S) = 0 \to \Lambda :=$ 'reject' and abort

$t_C \xleftarrow{\$} \mathbb{Z}_q, T_C := g^{t_C} \mod p, \sigma_C := \mathsf{SIG.Sign}(sk_C, m_1 || \ldots || m_8)$

$pms := T_S^{t_C} \mod p, ms := \mathsf{PRF}(pms, label_1 || r_C || r_S)$

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

$k := (k_{\mathsf{enc}}^{\mathsf{Client}} := (K_{\mathsf{enc}}^{C\to S}, K_{\mathsf{mac}}^{C\to S}), k_{\mathsf{dec}}^{\mathsf{Client}} := (K_{\mathsf{enc}}^{S\to C}, K_{\mathsf{mac}}^{S\to C}))$

$(m_7, m_8, m_9, m_{10}) := (cert_C, T_C, \sigma_C, flag_{enc})$

$\mathit{fin}_C := \mathsf{PRF}(ms, label_3 || m_1 || \ldots || m_{10})$

$$\xrightarrow{m_7, m_8, m_9, m_{10}}$$

$$\xrightarrow{m_{11} := \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, \mathit{fin}_C, st_e)}$$

$\Pi := C$, C is determined from $cert_C$

If $\mathsf{SIG.Vfy}(pk_\Pi, \sigma_C, m_1 || \ldots || m_8) = 0 \to \Lambda :=$ 'reject' and abort

$pms := T_C^{t_S} \mod p, ms := \mathsf{PRF}(pms, label_1 || r_C || r_S)$

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

$k := (k_{\mathsf{enc}}^{\mathsf{Server}} := (K_{\mathsf{enc}}^{S\to C}, K_{\mathsf{mac}}^{S\to C}), k_{\mathsf{dec}}^{\mathsf{Server}} := (K_{\mathsf{enc}}^{C\to S}, K_{\mathsf{mac}}^{C\to S}))$

$m_{12} := flag_{enc}, \mathit{fin}_S := \mathsf{PRF}(ms, label_4 || m_1 || \ldots || m_{12})$

$$\xleftarrow{m_{12}, m_{13} := \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, \mathit{fin}_S, st_e)}$$

If $\mathit{fin}_S \neq \mathsf{PRF}(ms, label_4 || m_1 || \ldots || m_{12}) \to \Lambda :=$ 'reject' and abort

else $\Lambda :=$ 'accept' and output k

If $\mathit{fin}_C \neq \mathsf{PRF}(ms, label_3 || m_1 || \ldots || m_{10}) \to \Lambda :=$ 'reject' and abort

pre-accept phase else $\Lambda :=$ 'accept' and output k

---

post-accept phase

$$\xRightarrow{\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, data, st_e)}$$

$$\xLeftarrow{\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, data, st_e)}$$

Fig. 2. Handshake protocol for ciphersuites TLS_DHE_* with client authentication

The `Server Hello` message m_2 has the same structure as `Client Hello`, with the only exception that at most one ciphersuite and one compression method can be present. Message m_3 may contain a certificate (or a chain of certificates, which is not considered in this paper) and the public key in the certificate must match the ciphersuite chosen by the server. For ephemeral Diffie-Hellman key exchange, the public key may be any key that can be used to sign messages. The Diffie-Hellman (DH) key exchange parameters are contained in the `Server Key Exchange` message m_4, including information on the DH group (e.g. prime number p and generator g for a prime-order q subgroup of $\mathbb{Z}_p^*$), the DH share T_S, and a signature computed over these values plus the two random numbers r_C and r_S. The next two messages are very simple: the `Certificate Request` message m_5 only contains a list of certificate types that the client may use to authenticate itself, and the `Server Hello Done` message m_6 does not contain any data, but consists only of a constant tag with byte-value '14' and a length value '0'.

Having received these messages, the signature σ_S is verified. If this fails, the client 'rejects' and aborts, otherwise the client completes the key exchange and computes the cryptographic keys. The `Client Certificate` message m_7 contains a signing certificate $cert_C$ with the public key pk_C of the client.[1] Message m_8 is called `Client Key Exchange`, and contains the Diffie-Hellman share T_C of the client. To authenticate the client, a signature σ_C is computed on a concatenation of all previous messages (up to m_8) and padded prefixes and sent in the `Certificate Verify` message m_9.

The client is now also able to compute the *premaster secret* pms, from which all further secret values are derived. 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, to derive all encryption and MAC keys as well as the `Client Finished` message fin_C. More precisely, the key material is computed as

$$K_{\mathsf{enc}}^{C \to S} || K_{\mathsf{enc}}^{S \to C} || K_{\mathsf{mac}}^{C \to S} || K_{\mathsf{mac}}^{S \to C} := \mathsf{PRF}(ms, label_2 || r_C || r_S). \tag{1}$$

After these computations have been completed, the keys are handed over to the TLS Record Layer of the client, which is now able to MAC and encrypt any data. To signal the 'start of encryption' to the server, a single message m_{10} (`Change Cipher Spec`) with byte value '1' ($flag_{enc}$) is sent unencrypted to S. Then message m_{11} consists of an authenticated encryption of the `Client Finished` message fin_C. After the server has received messages m_7, m_8, m_9, the server verifies the signature in m_9. If this fails, the server 'rejects' (i.e. sets $\Lambda =$ 'reject') and aborts. Otherwise it first determines pms and ms. From this the encryption and MAC keys are computed as in (1). It can then decrypt m_{11} and check fin_C by computing the pseudo-random value on the messages sent and received by the server. If this check fails, it 'rejects' and aborts. If the check is successful, it 'accepts' (i.e. sets $\Lambda =$ 'accept'), computes the `Server Finished`

[1] When either party receives a certificate $cert_X$, the partner id is set to $\Pi := X$.

message fin_S and sends messages m_{12} and m_{13} to the client. If the check of fin_S on the client side is successful, the client also 'accepts'.

The obtained keys can now be used to transmit payload data in the TLS Record Layer using a stateful symmetric encryption scheme $(\mathsf{StE.Enc}, \mathsf{StE.Dec})$.

Abbreviated TLS Handshakes, side-channels and cross-protocol attacks. In our analysis, we do not consider the abbreviated TLS Handshake, but note that the server can always enforce an execution of the full protocol. Moreover, we do not consider attacks based on side-channels, such as error messages, or cross-protocol attacks like [24].

4 AKE Protocols

While the established security models for, say, encryption (e.g. IND-CPA or IND-CCA security), or digital signatures (e.g., EUF-CMA), are clean and simple, a more complex model is required to model the capabilities of active adversaries to define secure authenticated key-exchange. An important line of research [6,10,19,13] dates back to Bellare and Rogaway [4], where an adversary is provided with an 'execution environment', which emulates the real-world capabilities of an active adversary, which has full control over the communication network. In the sequel we describe a variant of this model, which captures adaptive corruptions, perfect forward secrecy, and security against key-compromise impersonation attacks in a public-key setting.

Execution Environment. Consider a set of parties $\{P_1, \ldots, P_\ell\}$, $\ell \in \mathbb{N}$, where each party $P_i \in \{P_1, \ldots, P_\ell\}$ is a (potential) protocol participant and has a long-term key pair (pk_i, sk_i). To model several sequential and parallel executions of the protocol, each party P_i is modeled by a collection of oracles $\pi_i^1, \ldots, \pi_i^d$ for $d \in \mathbb{N}$. Each oracle π_i^s represents a process that executes one single instance of the protocol. All oracles $\pi_i^1, \ldots, \pi_i^d$ representing party P_i have access to the same long-term key pair (pk_i, sk_i) of P_i and to all public keys $pk_1, \ldots, pk_\ell$. Moreover, each oracle π_i^s maintains as internal state the following variables:

- $\Lambda \in \{\mathtt{accept}, \mathtt{reject}\}$.
- $k \in \mathcal{K}$, where $\mathcal{K}$ is the keyspace of the protocol.
- $\Pi \in \{1, \ldots, \ell\}$ containing the intended communication partner, i.e., an index j that points to a public key pk_j used to perform authentication.[2]
- Variable $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$.
- Some additional temporary state variable st (which may, for instance, be used to store ephemeral Diffie-Hellman exponents or a transcript of messages).

The internal state of each oracle is initialized to $(\Lambda, k, \Pi, \rho, st) = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$, where $V = \emptyset$ denotes that variable V is undefined. Furthermore, we will always

[2] We assume that each party P_i is uniquely identified by its public key pk_i. In practice, several keys may be assigned to one identity. Furthermore, there may be other ways to determine identities, for instance by using certificates. However, this is out of scope of this paper.

assume (for simplicity) that $k = \emptyset$ if an oracle has not reached $\texttt{accept}$-state (yet), and contains the computed key if an oracle is in $\texttt{accept}$-state, so that we have

$$k \neq \emptyset \iff \Lambda = \texttt{accept}. \tag{2}$$

An adversary may interact with these oracles by issuing the following queries.

– $\mathsf{Send}(\pi_i^s, m)$: The adversary can use this query to send message m to oracle π_i^s. The oracle will respond according to the protocol specification, depending on its internal state. If the attacker asks the first Send-query to oracle π_i^s, then the oracle checks whether $m = \top$ consists of a special 'initialization' symbol $\top$. If true, then it sets its internal variable $\rho := \mathsf{Client}$ and responds with the first protocol message. Otherwise it sets $\rho := \mathsf{Server}$ and responds as specified in the protocol. [3] The variables Λ, k, Π, st are also set after certain Send-queries. [4]

– $\mathsf{Reveal}(\pi_i^s)$: Oracle π_i^s responds to a Reveal-query with the contents of variable k. Note that we have $k \neq \emptyset$ if and only if $\Lambda = \texttt{accept}$, see (2).

– $\mathsf{Corrupt}(P_i)$: Oracle π_i^1 responds with the long-term secret key sk_i of party P_i.[5] If $\mathsf{Corrupt}(P_i)$ is the τ-th query issued by $\mathcal{A}$, then we say that P_i is *τ-corrupted.* For parties that are not corrupted we define $\tau := \infty$.

– $\mathsf{Test}(\pi_i^s)$: This query may be asked only once throughout the game. If π_i^s has state $\Lambda \neq \texttt{accept}$, then it returns some failure symbol $\bot$. Otherwise it flips a fair coin b, samples an independent key $k_0 \xleftarrow{\$} \mathcal{K}$, sets $k_1 = k$ to the 'real' key computed by π_i^s, and returns k_b.

Security Definition. Bellare and Rogaway [4] have introduced the notion of *matching conversations* in order to define correctness and security of an AKE protocol precisely. We denote with $T_{i,s}$ the sequence that consists of all messages sent and received by π_i^s in chronological order (not including the initialization-symbol $\top$). We also say that $T_{i,s}$ is the *transcript* of π_i^s. For two transcripts $T_{i,s}$ and $T_{j,t}$, we say that $T_{i,s}$ is a *prefix* of $T_{j,t}$, if $T_{i,s}$ contains at least one message, and the messages in $T_{i,s}$ are identical to and in the same order as the first $|T_{i,s}|$ messages of $T_{j,t}$.

Definition 3 (Matching conversations). *We say that π_i^s has a* matching conversation *to π_j^t, if*

– *$T_{j,t}$ is a prefix of $T_{i,s}$ and π_i^s has sent the last message(s), or*

– *$T_{i,s}$ is a prefix of $T_{j,t}$ and π_j^t has sent the last message(s).*

[3] Note that we assume that learning identities of communication partners (which is necessary to determine the public-key used to perform authentication) is part of the protocol.

[4] For details on when and how they are set in TLS, see the description in Section 3 and Figure 2.

[5] Note, that the adversary does not 'take control' of oracles corresponding to a corrupted party. But he learns the long-term secret key, and can henceforth simulate these oracles.

Security of AKE protocols is now defined by requiring that (i) the protocol is a secure authentication protocol, and (ii) the protocol is a secure key-exchange protocol.

AKE Game. We formally capture this notion as a game, played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The challenger implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. At the beginning of the game, the challenger generates ℓ long-term key pairs (pk_i, sk_i) for all $i \in [\ell]$. The adversary receives the public keys $pk_1, \ldots, pk_\ell$ as input. Now the adversary may start issuing Send, Reveal and Corrupt queries, as well as one Test-query. Finally, the adversary outputs a bit b' and terminates.

Definition 4. *We say that an adversary* (t, ϵ)-breaks *an AKE 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 leastA* ϵ *there exists an oracle* π_i^s *such that*
   - π_i^s *'accepts' when* $\mathcal{A}$ *issues its* τ_0*-th query with partner* $\Pi = j$*, and*
   - P_j *is* τ_j*-corrupted with* $\tau_0 < \tau_j$*,*[6] *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* $\mathcal{A}$ *issues a* Test*-query to any oracle* π_i^s *and*
   - $\mathcal{A}$ *does not issue a* Reveal*-query to* π_i^s*, nor to* π_j^t *such that* π_i^s *has a matching conversation to* π_j^t *(if such an oracle exists), and*
   - π_i^s *'accepts' when* $\mathcal{A}$ *issues its* τ_0*-th query, and both parties* P_i *and* P_j *are* τ_i*- and* τ_j*-corrupted, respectively, with* $\tau_0 < \tau_i, \tau_j$*,*[7]

   then the probability that $\mathcal{A}$ *outputs* b' *which equals the bit* b *sampled by the* Test*-query satisfies*
   $$|\Pr[b = b'] - 1/2| \geq \epsilon.$$

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

Remark 1. Note that the above definition even allows to corrupt oracles involved in the Test-session (of course only after the Test-oracle has reached accept-state, in order to exclude trivial attacks). Thus, protocols secure with respect to this definition provide *perfect forward secrecy.* Note also that we allow the 'accepting' oracle to be corrupted even *before* it reaches accept-state, which provides security against *key-compromise impersonation* attacks.

Now we can prove the security of a modified version of the TLS Handshake protocol. As discussed in the introduction, it is impossible to prove the full TLS Handshake protocol secure in any security model based on key-indistinguishability, like the model from Section 4, because the encryption and MAC of the

[6] That is, P_j is not corrupted (i.e. τ-corrupted with $\tau = \infty$) when π_i^s 'accepts'.

[7] That is, neither party P_i nor P_j is corrupted when π_i^s 'accepts'.

`Finished` messages provide a 'check value', that can be exploited by an adversary to determine the bit b chosen by the Test-query.

Therefore we consider a 'truncated TLS' protocol as in [20,21]. In this truncated version, we assume that the `Finished` messages are sent in clear, that is, neither encrypted nor authenticated by a MAC. More precisely, we modify the TLS protocol depicted in Figure 2 such that messages m_{11} and m_{13} contain only fin_{Π} (instead of $\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\Pi}, \mathsf{len}, H, \mathit{fin}_{\Pi}, st_e)$), allowing us to prove security in the above model.

Theorem 1. *Let μ be the output length of* PRF *and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}})$-secure, the signature scheme is $(t, \epsilon_{\mathsf{sig}})$-secure, the* DDH*-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, and the* $\mathsf{PRF\text{-}ODH}$*-problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and* PRF.

Then for any adversary that $(t', \epsilon_{\mathsf{ttls}})$-breaks the truncated TLS-DHE protocol in the sense of Definition 4 with $t \approx t'$ holds that

$$\epsilon_{\mathsf{ttls}} \leq 4 \cdot d\ell \left(\frac{d\ell}{2^{\lambda}} + \ell \cdot \epsilon_{\mathsf{sig}} + \frac{5}{4} \cdot \epsilon_{\mathsf{ddh}} + \frac{5}{2} \cdot \epsilon_{\mathsf{prf}} + d\ell \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^{\mu}} \right) \right).$$

Proof Sketch. Let us sketch the proof of Theorem 1, more details can be found in the full version [18]. We consider three types of adversaries:

1. Adversaries that succeed in making an oracle accept maliciously, such that the first oracle that does so is a Client-oracle (i.e., an oracle with $\rho = \mathsf{Client}$). We call such an adversary a Client-adversary.
2. Adversaries that succeed in making an oracle accept maliciously, such that the first oracle that does so is a Server-oracle (i.e., an oracle with $\rho = \mathsf{Server}$). We call such an adversary a Server-adversary.
3. Adversaries that do not succeed in making any oracle accept maliciously, but which answer the Test-challenge. We call such an adversary a Test-adversary.

We prove Theorem 1 by three lemmas. Lemma 1 bounds the probability $\epsilon_{\mathsf{client}}$ that a Client-adversary succeeds, Lemma 2 bounds the probability $\epsilon_{\mathsf{server}}$ that a Server-adversary succeeds, and Lemma 3 bounds the success probability ϵ_{ke} of a Test-adversary. Then we have $\epsilon_{\mathsf{ttls}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + \epsilon_{\mathsf{ke}}$.

Lemma 1. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s with $\rho =$* Client *that accepts maliciously is at most*

$$\epsilon_{\mathsf{client}} \leq d\ell \left(\frac{d\ell}{2^{\lambda}} + \ell \cdot \epsilon_{\mathsf{sig}} + d\ell \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^{\mu}} \right) \right)$$

where all quantities are defined as stated in Theorem 1.

Proof Sketch. We prove Lemma 1 in a *sequence of games* [5,23].

Game 0. This is the original security experiment.

Game 1. We add an abort condition. The challenger aborts, if throughout the game any oracle chooses a random nonce r_C or r_S which is not unique. Since nonces are chosen uniformly random, the collision probability is bounded by $(d\ell)^2 2^{-\mu}$. This abort condition ensures that any oracle that accepts with non-corrupted partner has a *unique* partner oracle.

Game 2. The challenger guesses an oracle $\pi_{i^*}^{s^*}$, and aborts if this oracle does not accept maliciously with $\rho = \mathsf{Client}$. If there exists a maliciously accepting Client-oracle, then the guess is correct with probability $1/d\ell$.

Game 3. Next we want to ensure that $\pi_{i^*}^{s^*}$ receives as input exactly the Diffie-Hellman share T_S chosen by another oracle π_j^t (not by the adversary). Note that the respective party P_j must not be corrupted, as otherwise $\pi_{i^*}^{s^*}$ would not accept maliciously in the sense of Definition 4. The Diffie-Hellman share T_S is contained in the digital signature received by $\pi_{i^*}^{s^*}$, thus we can use the $(\epsilon_{\mathsf{sig}}, t)$-security of the signature scheme to ensure that the adversary can only forward T_S from π_j^t to $\pi_{i^*}^{s^*}$.

Game 4. In this game the challenger guesses upfront the oracle $\pi_{j^*}^{t^*}$ that chooses and signs the Diffie-Hellman share T_S received by $\pi_{i^*}^{s^*}$, and aborts if its guess is wrong. Again the guess is correct with probability at least $1/d\ell$.

Game 5. Now we are in a game where the challenger controls both Diffie-Hellman shares $T_C = g^{t_c}$ and $T_S = g^{t_s}$ chosen and received by $\pi_{i^*}^{s^*}$. A natural approach would be to use the DDH assumption now to replace the premaster-secret $pms = g^{t_c t_s}$ with an independent random value $\widetilde{pms}$, in order to be able to use the security of the $\mathsf{PRF}(\widetilde{pms}, \cdot)$ as an argument in a following game to replace the master secret ms with an independent $\widetilde{ms}$. However, unfortunately we cannot do this, as this would lead to a problem with the simulation of the fin_S-message sent by $\pi_{j^*}^{t^*}$ (we describe this issue in more detail in Section 6). Instead, we use the $\mathsf{PRF\text{-}ODH}$-assumption to directly replace the master secret ms with an independent value $\widetilde{ms}$. We use the oracle provided by the $\mathsf{PRF\text{-}ODH}$-assumption to simulate the fin_S message if necessary, which allows us to overcome the mentioned problem.

Game 6. In this game the challenger replaces the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ with a truly random function. Note that $\widetilde{ms}$ is an independent random value, thus we can use the security of the PRF to argue that this game is indistinguishable from Game 5.

Game 7. Finally, we use the fact that in this game a truly random function is used to verify the finished-message fin_S received by $\pi_{i^*}^{s^*}$, to conclude that $\pi_{i^*}^{s^*}$ accepts maliciously with probability at most $2^{-\mu}$.

Lemma 2. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s with $\rho = \mathsf{Server}$ that accepts maliciously is at most*

$$\epsilon_{\mathsf{server}} \le d\ell \left(\frac{d\ell}{2^\lambda} + \ell \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \frac{1}{2^\mu} \right)$$

where all quantities are defined as stated in Theorem 1.

Proof Sketch. The proof of Lemma 2 is very similar to the proof of Lemma 2, except that the problem with the simulation of the fin_S message does not occur in the case where we are dealing with Server-adversaries. Therefore we are able to base security in this case on the standard DDH assumption instead of the non-standard $\mathsf{PRF\text{-}ODH}$ assumption. (This is the reason why we consider Client- and Server-adversaries separately).

Lemma 3. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the* Test*-challenge correctly is at most* $1/2 + \epsilon_{\mathsf{ke}}$ *with*

$$\epsilon_{\mathsf{ke}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + d\ell \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) .$$

where $\epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 4 (cf. Lemmas 1 and 2) and all other quantities are defined as stated in Theorem 1.

Proof Sketch. In order to prove Lemma 3, we first use the bounds derived in Lemmas 1 and 2 on the probability that there exists a Client- or Server-oracle that accepts maliciously. We then employ a very similar sequence of games as in the proofs of Lemmas 1 and 2. Recall that the keys in the real protocol are computed as

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

and in the proofs of Lemmas 1 and 2 we have first replaced ms with an independent value $\widetilde{ms}$, and then the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ with a truly random function. Once we have reached this game, the adversary will *always* receive an independent key vector $K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C}$ as input, regardless of the bit b sampled for the Test-query. Thus, the adversary outputs its guess b' without receiving any information about b. This allows us to bound the success probability of the adversary in this final game as $\Pr[b' = b] = 1/2$.

Summing up probabilities from Lemmas 1 to 3, we obtain that

$$\begin{aligned}
\epsilon_{\mathsf{ttls}} \leq &\epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + \epsilon_{\mathsf{ke}} \leq 2 \cdot (\epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}) + d\ell \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) \\
\leq &4 \cdot \max\{\epsilon_{\mathsf{client}}, \epsilon_{\mathsf{server}}\} + d\ell \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) \\
\leq &4 \cdot d\ell \left(\frac{d\ell}{2^\lambda} + \ell \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + d\ell \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^\mu}\right)\right) \\
&+ d\ell \left(\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}\right) \\
= &4 \cdot d\ell \left(\frac{d\ell}{2^\lambda} + \ell \cdot \epsilon_{\mathsf{sig}} + \frac{5}{4} \cdot \epsilon_{\mathsf{ddh}} + \frac{5}{2} \cdot \epsilon_{\mathsf{prf}} + d\ell \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^\mu}\right)\right) .
\end{aligned}$$

5 ACCE Protocols

An *authenticated and confidential channel establishment* (ACCE) protocol is a protocol executed between two parties. The protocol consists of two phases, called the 'pre-accept' phase and the 'post-accept' phase.

Pre-accept phase. In this phase a 'handshake protocol' is executed. In terms of functionality this protocol is an AKE protocol as in Section 4, that is, both communication partners are mutually authenticated, and a session key k is established. However, it need not necessarily meet the security definition for AKE protocols (Definition 4). This phase ends, when both communication partners reach an $\texttt{accept}$-state.

Post-accept phase. This phase is entered, when both communication partners reach $\texttt{accept}$-state. In this phase data can be transmitted, encrypted and authenticated with key k.

The prime example for an ACCE protocol is TLS. Here, the pre-accept phase consists of the TLS Handshake protocol. In the post-accept phase encrypted and authenticated data is transmitted over the TLS Record Layer.

To define security of ACCE protocols, we combine the security model for authenticated key exchange from Section 4 with stateful length-hiding encryption in the sense of [22]. Technically, we provide a slightly modified execution environment that extends the types of queries an adversary may issue.

Execution environment. The execution environment is very similar to the model from Section 4, except for a few simple modifications. We extend the model such that in the post-accept phase an adversary is also able to 'inject' chosen-plaintexts by making an $\mathsf{Encrypt}$-query, and chosen-ciphertexts by making a $\mathsf{Decrypt}$-query. Moreover, each oracle π_i^s keeps as additional internal state a (randomly chosen) bit $b_i^s \xleftarrow{\$} \{0,1\}$, two counters u and v required for the security definition, and two state variables st_e and st_d for encryption and decryption with a stateful symmetric cipher. In the sequel we will furthermore assume that the key k consists of two different keys $k = (k_{\mathsf{enc}}^{\rho}, k_{\mathsf{dec}}^{\rho})$ for encryption and decryption. Their order depends on the role $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$ of oracle π_i^s. This is the case for TLS (see Section 3).

An adversary issue the following queries to the provided oracles.

- $\mathsf{Send}^{\mathsf{pre}}(\pi_i^s, m)$: This query is identical to the Send-query in the AKE model from Section 4, except that it replies with an error symbol $\perp$ if oracle π_i^s has state $\Lambda = \texttt{accept}$. ($\mathsf{Send}$-queries in accept-state are handled by the $\mathsf{Decrypt}$-query below).
- $\mathsf{Reveal}(\pi_i^s)$ and $\mathsf{Corrupt}(P_i)$: These queries are identical to the corresponding queries in the AKE model from Section 4.
- $\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)$: This query takes as input two messages m_0 and m_1, length parameter len, and header data H. If $\Lambda \neq \texttt{accept}$ then π_i^s returns $\perp$. Otherwise, it proceeds as $\mathsf{Encrypt}(m_0, m_1, \mathsf{len}, H)$ depicted in Figure 1 with $k = k_{\mathsf{enc}}^{\rho}$ and $b = b_i^s$ depending on the internal state of π_i^s.
- $\mathsf{Decrypt}(\pi_i^s, C, H)$: This query takes as input a ciphertext C and header data H. If $\Lambda \neq \texttt{accept}$ then π_j^t returns $\perp$. Otherwise, it proceeds as $\mathsf{Decrypt}(C, H)$ depicted in Figure 1 with $k = k_{\mathsf{dec}}^{\rho}$ and $b = b_i^s$ depending on the internal state of π_i^s.

Security Definition. Security of ACCE protocols is defined by requiring that (i) the protocol is a secure authentication protocol (ii) in the post-accept phase data is transmitted over an authenticated and confidential channel in the sense of Definition 2.

Again this notion is captured by a game, played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The challenger implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. At the beginning of the game, the challenger generates ℓ long-term key pairs (pk_i, sk_i) for all $i \in [\ell]$. The adversary receives the public keys $pk_1, \ldots, pk_\ell$ as input. Now the adversary may start issuing Send, Reveal, Corrupt, Encrypt, and Decrypt queries. Finally, the adversary outputs a triple (i, s, b') and terminates.

Definition 5. *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' when $\mathcal{A}$ issues its τ_0-th query with partner $\Pi = j$, and*
   - *P_j is τ_j-corrupted with $\tau_0 < \tau_j$,[8] and*
   - *there is no unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t.*
2. *When $\mathcal{A}$ terminates and outputs a triple (i, s, b') such that*
   - *π_i^s 'accepts' when $\mathcal{A}$ issues its τ_0-th query with intended partner $\Pi = j$, and P_j is τ_j-corrupted with $\tau_0 < \tau_j$,*
   - *$\mathcal{A}$ did not issue a* Reveal-*query to π_i^s, nor to π_j^t such that π_i^s has a matching conversation to π_j^t (if such an oracle exists), and*

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

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

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 $\mathcal{A}$ anwers the encryption-challenge correctly.

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

Remark 2. Note that the above definition even allows to corrupt the oracle π_i^s whose internal secret bit the attacker tries to determine. Of course this is only allowed after π_i^s has reached `accept`-state, in order to exclude trivial attacks. Thus, protocols secure with respect to this definition provide *perfect forward secrecy.* Note also that again we allow the 'accepting' oracle to be corrupted even *before* it reaches `accept`-state, which provides security against *key-compromise impersonation* attacks.

Relation to the AKE Security Definition from Section 4. Note that an ACCE protocol can be constructed in a two-step approach.

1. (AKE part) First an authenticated key-exchange (AKE) protocol is executed. This protocol guarantees the authenticity of the communication partner, and provides a cryptographically 'good' (i.e., for the attacker indistinguishable from random) session key.

[8] That is, P_j is not corrupted (i.e. τ-corrupted with $\tau = \infty$) when π_i^s 'accepts'.

2. (Symmetric part) The session key is then used in a symmetric encryption scheme providing integrity and confidentiality.

This modular approach is simple and generic, and therefore appealing. It can be shown formally that this two-step approach yields a secure ACCE protocol, if the 'AKE part' meets the security in the sense of Definition 4, and the 'symmetric part' consists of a suitable authenticated symmetric encryption scheme (e.g. secure according to Definition 2).

However, if the purpose of the protocol is the establishment of an authenticated confidential channel, then it is not necessary that the 'AKE-part' of the protocol provides full indistinguishability of *session keys*. It actually would suffice if *encrypted messages* are indistinguishable, and *cannot be altered* by an adversary. These requirements are strictly weaker than indistinguishability of keys in the sense of Definition 4, and thus easier to achieve (possibly from weaker hardness assumptions, or by more efficient protocols).

We stress that our ACCE definition is mainly motivated by the fact that security models based on key indistinguishability do not allow for a security analysis of full TLS, as detailed in the introduction. We do not want to propose ACCE as a new security notion for key exchange protocols, since it is very complex and the modular two-step approach approach seems more useful in general.

Theorem 2. *Let μ be the output length of* PRF *and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}})$-secure, the signature scheme is $(t, \epsilon_{\mathsf{sig}})$-secure, the* DDH*-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, and the* PRF-ODH*-problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and* PRF*. Suppose that the stateful symmetric encryption scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure.*

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-DHE protocol in the sense of Definition 5 with $t \approx t'$ holds that

$$\epsilon_{\mathsf{tls}} \leq 4d\ell \left(\frac{d\ell}{2^{\lambda}} + \ell\epsilon_{\mathsf{sig}} + \frac{5}{4}\epsilon_{\mathsf{ddh}} + \frac{5}{2}\epsilon_{\mathsf{prf}} + \frac{1}{4}\epsilon_{\mathsf{sLHAE}} + d\ell \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^{\mu}} \right) \right) .$$

Proof Sketch. The proof of Theorem 2 is very similar to the proof of Theorem 1. Instead of proving indistinguishability of keys, as in Lemma 3 from the proof of Theorem 1, we now have to consider indistinguishability of encrypted messages and authenticity of ciphertexts. We do this by employing the same sequence of games as in the proof of Lemma 3, except that we extend the proof by one game-hop at the end of the sequence of games.

Recall that in the proof of Lemma 3 the keys $K_{\mathsf{enc}}^{C \to S} || K_{\mathsf{enc}}^{S \to C} || K_{\mathsf{mac}}^{C \to S} || K_{\mathsf{mac}}^{S \to C}$, which determine the encryption and decryption keys $(k_{\mathsf{enc}}^{\rho}, k_{\mathsf{dec}}^{\rho})$ of the stateful encryption scheme, were replaced with independent random values. This allows us to extend the sequence of games by one final game, where the security of TLS-DHE is reduced to the sLHAE-security (in the sense of Definition 2) of the underlying stateful encryption scheme.

6 Security of TLS-DHE from Standard Assumptions

In this section we sketch why we had to make the $\mathsf{PRF\text{-}ODH}$-assumption in the proof of Lemma 1 (and thus in Theorems 1 and 2), and why it seems unlikely that one can prove security based on standard DDH and a standard assumption on the PRF, if a security model allowing active adversaries and user corruptions is considerd.

Suppose we are given a Client-adversary, that is, an adversary which always makes Client-oracle $C := \pi_i^s$ (i.e., π_i^s with $\rho = \mathsf{Client}$) accept maliciously with intended partner $\Pi = S$. Suppose we want to argue that the adversary is not able to forge the fin_S-message received by C (which we would have to, since the fin_S-message is the only message that cryptographically protects all messages previously received by π_i^s, and thus is required to ensure that π_i^s has a matching conversation to some other oracle), and that we want to assume only that the PRF is secure in the standard sense (see [18, Definition 3]). Then at some point in the proof we would have to replace the premaster secret computed by π_i^s as $pms = T_S^{t_C} = g^{t_C t_S}$ with an independent random value.

Note that in order to do so and to argue in the proof with indistinguishability, we must not know any of the exponents t_C and t_S in $T_C = g^{t_C}$ and $T_S = g^{t_S}$, as otherwise we can trivially distinguish the real $pms = g^{t_C t_S}$ from a random pms'. The problematic property of TLS-DHE is now, that an adversary may test whether the challenger 'knows' t_S, and then make Client-oracle π_i^s accept maliciously only if this holds. This works as follows.

1. The adversary establishes a communication between two oracles π_i^s (representing the client C) and π_j^t (representing the server S) by simply forwarding the messages m_1 and $(m_2, \ldots, m_6)$ between C and S.
2. Then C will respond with $(m_7, \ldots, m_{11}) = (\mathit{cert}_C, T_C, \sigma_C, \mathit{flag}_{enc}, \mathit{fin}_C)$.[9] This message is not forwarded.
3. Instead, the adversary corrupts some party $P^* \notin \{P_i, P_j\}$, and obtains the secret key sk^* of this party. Then it computes
   (a) $T^* := g^{t^*} \bmod p$ for random $t^* \xleftarrow{\$} \mathbb{Z}_q$,
   (b) $\sigma^* := \mathsf{SIG.Sign}(sk^*; (r_C, r_S, T_S, T^*))$ using the corrupted key sk^*,
   (c) $ms^* := \mathsf{PRF}(T_S^{t^*}, \mathit{label}_1 || r_C || r_S)$ using knowledge of t^*, and
   (d) $\mathit{fin}_C^* := \mathsf{PRF}(ms^*, m_1 || m_2 || (T^*, \sigma^*))$.

   and sends $(m_7, \ldots, m_{11}) = (\mathit{cert}_{C^*}, T^*, \sigma^*, \mathit{flag}_{enc}, \mathit{fin}_C^*)$ to S. Note that S cannot determine that its communication partner has changed, because any messages previously received by S were perfectly anonymous.
4. If S responds with a correct fin_S^*-message (note that the adversary is able to compute the key $pms^* := T_S^{t^*}$, since it 'knows' t^*, and thus is able to verify the validity of fin_S^*), then adversary concludes that the challenger 'knows' t_S and forges the required fin_S-message to make π_i^s accept without matching conversations. Otherwise the adversary aborts.

[9] We consider truncated TLS-DHE here for simplicity, the same argument applies to TLS-DHE with encrypted `Finished`-messages, too.

Note that the above adversary is a valid, successful adversary in the real security experiment. It does not issue any Reveal-query and only one $\mathsf{Corrupt}$-query to an unrelated party, such that the intended communication partner $\Pi = S$ of $C = \pi_i^s$ remains uncorrupted, but still it makes $C = \pi_i^s$ 'accept' and there is no oracle that C has a matching conversation to.

However, we explain why we will not be able to use this adversary in a simulated security experiment, where the challenger does not know the exponent t_S of $T_S = g^{t_S}$. Intuitively, the reason is that in this case the challenger would *first* have to compute the `Finished`-message fin_S^*, where

$$\mathit{fin}_S^* = \mathsf{PRF}(ms, m_1||\ldots||m_3) \quad \text{and} \quad ms = \mathsf{PRF}(T_S^{t^*}, \mathit{label}_1||r_C||r_S),$$

but 'knowing' neither $t_S = \log T_S$, nor $t^* = \log T^*$. This is the technical problem we are faced with, if we want to prove security under a standard assumption like DDH. Under the $\mathsf{PRF\text{-}ODH}$-assumption, we can however use the given oracle to compute first ms, and from this the `Finished`-message fin_S^*.

Interestingly, the above technical problem does not appear if we consider only Server-adversaries (i.e., adversaries that make an oracle π_i^s accept maliciously with $\rho = \mathsf{Server}$) instead. This is due to an asymmetry of the TLS-DHE Handshake protocol, see [18] for details.

One can circumvent the above problem, and thus base the security proof on the standard DDH assumption instead of $\mathsf{PRF\text{-}ODH}$, if one considers a weaker security model where no $\mathsf{Corrupt}$-queries are allowed (which however seems not adequate for the way how TLS-DHE is used on the Internet).

In [11] Canetti and Krawczyk describe a protocol called Σ_0, which exhibits many similarities to the TLS-DHE Handshake protocol, but is provably secure under standard assumptions (in particular under DDH instead of $\mathsf{PRF\text{-}ODH}$). We discuss why the subtle differences between Σ_0 and TLS-DHE are crucial in [18, Section 8]. We also note that one could, *in principle*, make TLS-DHE provably secure under standard assumptions, if one would modify it such that it becomes more similar to Σ_0, which would allow to carry the security analysis of Σ_0 from [11] over to TLS-DHE. Of course it seems unrealistic that such substantial changes become accepted in practice.

7 Conclusion

We have shown that the core cryptographic protocol underlying TLS-DHE provides a secure establishment of confidential and authenticated channels. We can avoid the random oracle model, if we make a suitable assumption on the pseudorandom function. The goal of this work is to analyse TLS-DHE on the protocol layer. As common in cryptographic protocol analyses, we therefore have ignored implementational issues like error messages, which of course might also be used to break the security of the protocol. We leave it as an interesting open question to find an adequate approach for modeling such side-channels in complex scenarios like AKE involving many parties and parallel, sequential, and concurrent executions.

The whole TLS protocol suite is much more complex than the cryptographic protocol underlying TLS-DHE. It is very flexible, as it allows to negotiate ciphersuites at the beginning of the protocol, or to resume sessions using an abbreviated TLS Handshake. So clearly the security analysis of TLS is not finished yet, there are still many open questions. However, we consider this work as a strong indicator for the soundness of the TLS protocol framework. We believe that future revisions of the TLS standard should be guided by provable security – ideally in the standard model.

Acknowledgements. We would like to thank Dennis Hofheinz, Kenny Paterson, Zheng Yang, and the anonymous referees for helpful comments and discussions.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (2001)
2. Langley, A., Google Security Team: Protecting data for the long term with forward secrecy, `http://googleonlinesecurity.blogspot.co.uk/2011/11/protecting-data-for-long-term-with.html`
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
5. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
6. Blake-Wilson, S., Johnson, D., Menezes, A.: Key Agreement Protocols and their Security Analysis. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (1997)
7. Bleichenbacher, D.: Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998)
8. Brzuska, C., Fischlin, M., Smart, N.P., Warinschi, B., Williams, S.: Less is more: Relaxed yet composable security notions for key exchange. Cryptology ePrint Archive, Report 2012/242 (2012), `http://eprint.iacr.org/`
9. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, pp. 136–145. IEEE Computer Society Press (October 2001)
10. Canetti, R., Krawczyk, H.: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
11. Canetti, R., Krawczyk, H.: Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (2002), `http://eprint.iacr.org/2002/120/`

12. Choo, K.-K.R., Boyd, C., Hitchcock, Y.: Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 585–604. Springer, Heidelberg (2005)
13. Cremers, C.J.F.: Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 20–33. Springer, Heidelberg (2009)
14. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard) (January 1999); Obsoleted by RFC 4346, updated by RFCs 3546, 5746
15. Dierks, T., Rescorla, E.: 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
16. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (August 2008); Updated by RFCs 5746, 5878
17. Gajek, S., Manulis, M., Pereira, O., Sadeghi, A.-R., Schwenk, J.: Universally Composable Security Analysis of TLS. In: Baek, J., Bao, F., Chen, K., Lai, X. (eds.) ProvSec 2008. LNCS, vol. 5324, pp. 313–327. Springer, Heidelberg (2008)
18. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the Standard Model (full version). Cryptology ePrint Archive, Report 2011/219 (2011) (revised 2012), `http://eprint.iacr.org/2011/219`
19. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger Security of Authenticated Key Exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
20. Morrissey, P., Smart, N.P., Warinschi, B.: A Modular Security Analysis of the TLS Handshake Protocol. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 55–73. Springer, Heidelberg (2008)
21. Morrissey, P., Smart, N.P., Warinschi, B.: The TLS Handshake protocol: A modular analysis. Journal of Cryptology 23(2), 187–223 (2010)
22. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag Size *Does* Matter: Attacks and Proofs for the TLS Record Protocol. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 372–389. Springer, Heidelberg (2011)
23. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (November 2004)
24. Wagner, D., Schneier, B.: Analysis of the SSL 3.0 protocol. In: Proceedings of the Second USENIX Workshop on Electronic Commerce, pp. 29–40. USENIX Association (1996)