On the Security of TLS-DHE in the Standard Model

Tibor Jager
Karlsruhe Institute of Technology
Germany
tibor.jager@kit.edu

Florian Kohlar
Horst Görtz Institute for IT Security
Bochum, Germany
florian.kohlar@rub.de

Sven Schäge[1]
University College London
United Kingdom
s.schage@ucl.ac.uk

Jörg Schwenk
Horst Görtz Institute for IT Security
Bochum, Germany
joerg.schwenk@rub.de

June 27, 2012

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 in any classical key-indistinguishability-based security model (like e.g. the Bellare-Rogaway or the Canetti-Krawczyk model), due to subtle issues with the encryption of the final `Finished` messages of the TLS Handshake. Therefore we start with proving the security of a truncated version of the TLS Handshake protocol, which has also been considered in previous work 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 protocol with the TLS Record Layer can be proven secure in this model.

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

[1] Supported by EPSRC grant number EP/G013829/1.

Contents

1 Introduction

TRANSPORT LAYER SECURITY (TLS) is the single most important Internet security mechanism today. Due to a subtle interleaving of the TLS Handshake protocol with the data encryption in the TLS Record Layer it is impossible to prove the security of TLS using well-established security models [11, 19, 18] which define security via indistinguishability of keys. Therefore there is no security proof for the complete protocol up to now, as we illustrate below. Instead, all prior work either considered a modified version of the TLS Handshake [34, 45], or weaker security goals [32].

In this paper 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 Handshake protocol, which has been subject to prior work on TLS [34, 45], is an authenticated key exchange protocol in a security model that extends the Bellare-Rogaway model [11] to the public-key setting with adaptive corruptions and perfect forward secrecy (cf. [14]). Then we extend both the model and the proof to cover the combination of the TLS Handshake protocol with the TLS Record Layer, which allows us to show the security of the cryptographic core of a full TLS ciphersuite.

In our analysis we assume that the majority of building blocks of TLS-DHE (digital signature scheme, Diffie-Hellman key exchange, symmetric cipher) meets standard security properties. Solely for the pseudo-random function we require an additional non-standard security assumption, which is a variant of the Oracle Diffie-Hellman assumption [1]. We also explain why such a non-standard assumptions seems hard to avoid, if a security model with corruptions is considered. Our proofs are stated for mutual authentication, i.e., the client authenticates itself using a client certificate.

PROVING SECURITY OF TLS. The full TLS Handshake does not provide indistinguishable keys due to an interleaving of the key exchange part of TLS (the TLS Handshake protocol) and the data encryption in the TLS Record Layer. This interleaving provides a 'check value' that allows to test whether a given key is 'real' or 'random'. More precisely, the final messages of the TLS Handshake protocol (the `Finished` messages), which are essential to provide security against active adversaries like e.g. man-in-the-middle attackers, are first prepended with constant byte values (which provides us with known plaintext), then integrity protected by a MAC (which is instantiated with a pseudo-random function) and encrypted with the keys obtained from the TLS Handshake protocol. Thus, whenever an adversary receives a challenge key in response to a Test query, he can try to decrypt the `Finished` message and check validity of the MAC. If this succeeds, he will output 'real', and otherwise 'random'. Even changing the behavior of the Test query to only return the decryption keys (and not the MAC keys) does not help, since the adversary could still use the known plaintext bytes to answer the Test query successfully. Therefore it is impossible to prove the full TLS Handshake protocol secure in any security model based on indistinguishability of keys. Morissey *et al.* [45] have thereofre introduced a truncated TLS Handshake protocol, where the final encryption of the `Finished` messages is omitted.

In addition, in this paper we indentify another subtle issue, makes it hard to prove TLS-DHE secure under standard assumptions, if a security model is considered which allows the adversary to corrupt users and no additional non-standard assumption on the pseudo-random function PRF used in TLS is made.

1.1 Contribution

The paradox that the most important AKE protocol cannot be proven secure in any existing security model can be solved in two ways. Either one considers a truncated version of the TLS Handshake by omitting the encryption of the two `Finished` messages, or a new model for the combination of the TLS Handshake

protocol and the Application data protocol must be devised. In this paper we follow both approaches.

First we give a security proof for the truncated version of the TLS-DHE Handshake protocol. This allows to compare our results to previous work. The proof relies on the DDH assumption, an additional assumption called PRF-ODH that we need due to the issue discussed above, and the assumption that the building blocks of TLS (i.e. the signature scheme and the pseudo-random function) have certain security properties. It remains to analyse whether the building blocks have the required properties. Here we can partially build on previous work that analysed particular TLS components, see Section 1.2 for details.

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'[1] in the sense of stateful length-hiding authenticated encryption [48]. 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 core of the *full* TLS-DHE ciphersuites, i.e., the combination of the TLS Handshake with the TLS Record Layer, forms a secure ACCE protocol, if the Record Layer provides security in the sense of length-hiding authenticated encryption. Note that the latter was proven recently by Paterson *et al.* [48] for CBC-based Record Layer protocols.

Finally, we discuss the subtle property of TLS-DHE, which seems to make it hard to prove security without making an additional non-standard assumption, like PRF-ODH, on the PRF, in our security model. We also discuss several options to obtain a security proof under standard assumptions, which include considering a weak security model that disallows corruptions, and modifications to TLS-DHE.

PRACTICAL IMPACT. We stress that in practice TLS-DHE is much less used than TLS with encrypted key transport using an RSA-based encryption scheme (TLS-RSA). At the same time, mutual authentication is rarely used in practice. Typically, TLS is first used to authenticate the server and to establish a 'secure communication channel' between client and server. In the next step, the client authenticates itself by sending his authentication information over this secure communication channel to the server.

We believe that our result is nevertheless of high practical value. 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 with encrypted key transport 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, in our analysis we can show that TLS-DHE provides *perfect forward secrecy* – a very strong form of long-term 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 encryptions of session keys. 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 to provide (and push) perfect forward secrecy by using TLS-DHE as the default key exchange method for its services [2].

[1] Not to be confused with 'secure channels' in the sense of [19].

Third, it is likely that giving a security proof of the actually most widespread TLS Handshake 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 [15]. This problem was circumvented in previous work by either using an abstract public-key encryption scheme which is secure against chosen-ciphertext attacks [45], or by assuming PKCS#1 v2.0 (RSA-OAEP), which is not used in TLS, and omitting authentication [32].

INTERPRETATION. Our results show that the core cryptographic protocol of TLS is cryptographically sound, if the building blocks are suitably secure. By combining our work with [48] we obtain a standard-model security proof of core TLS 1.1 and 1.2 for *current* ciphersuites if we assume directly that the signature scheme is EUF-CMA secure.[2]

Our results 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.

CHOICE OF SECURITY MODEL. Authenticated key exchange is a basic building block in modern cryptography. However, since many different security models for different purposes exist [10, 11, 14, 18, 19, 24, 41, 22], 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 [11]. Variants of this model have also been studied by [22, 14], and especially by [45]. Detailed reasons for our choice are given in Appendix A.

1.2 Security Requirements on TLS Building Blocks

In our proofs we reduce the security of ephemeral Diffie-Hellman ciphersuites to certain security properties of building blocks of TLS. These building blocks (see Section 3 for precise definitions) are:

PSEUDORANDOM FUNCTION. For the pseudo-random function used in TLS we essentially require that the pseudo-random function (i) is secure in the standard sense (see Definition 3) and (ii) meets an additional requirement. The additional requirement on the PRF is related to the Oracle Diffie-Hellman (ODH) assumption, as introduced by Abdallah, Bellare, and Rogaway in 2001 to prove security of DHIES [1]. Although the PRF-ODH assumption is non-standard, it is, in a way, the best we can achieve. We argue in Section 8 why we need this assumption, if we consider a security model that allows the attacker to corrupt parties. We also remark in Section 8 that we can obtain a proof under the DDH assumption instead of PRF-ODH, if we (a) either do not allow corruptions or (b) slightly modify the TLS Handshake. However, although the latter idea might guide future revisions of the TLS standard, it is not an option for the current work in which we are concerned with the security of the present TLS protocol. Disallowing corruptions on the other hand would make the security model unrealistically weak.

Besides this non standard assumption, our proofs require that the pseudo-random function PRF meets the standard security definition for pseudo-random functions. All TLS versions specify a construction of PRF from cryptographic hash functions. TLS 1.2 prescribes SHA-256 [29], while previous standards used MD5 [51] and SHA-1 [30]. Foque *et al.* [31] where able to show that the pseudo-random function of

[2] To our best knowledge there is no security proof for the currently used schemes, but also no result contradicting this assumption.

TLS 1.2 constitutes a secure randomness extractor for *two* different key spaces simultaneously (albeit under different security assumptions, which however all are related to the fact that the compression function of the underlying hash function behaves like a pseudo-random function) – the key may either be a random bit-string, or a random element of a prime-order group (either a group defined over an elliptic curve or a *subgroup* of Z_p^*). Their work focuses on TLS 1.2, while stressing that the implementation of the key derivation function is not very different from the previous standards. We believe that similar results can easily be obtained for TLS 1.0 and TLS 1.1.

SYMMETRIC ENCRYPTION. The purpose of the TLS protocol is to establish an authenticated symmetric secret between two parties first (in the TLS Handshake), and then to use this secret to provide a 'secure communication channel' based on symmetric encryption (in the TLS Record Layer). While the informal idea of a 'secure communication channel' is simple, defining its security requirements precisely is not so trivial.

For instance, it is well-known that using IND-CCA secure encryption in the Record Layer is not sufficient to provide what is expected from a secure TLS channel, since it does not prevent many relevant attacks. For instance, it does not rule out replaying, dropping or re-ordering of ciphertexts. This is certainly not desirable in a 'secure communication channel', since it may lead to various kinds of attacks (cf. [16]). This issue can be solved by using a suitable *stateful* encryption scheme [6, 7]. For instance, TLS uses a 'MAC-then-Encode-then-Encrypt' (MEE) approach where a sequence counter is included in the MAC of each ciphertext. Moreover, it is well-known that sometimes even only the plaintext *length* may reveal valuable informations to an adversary, such as web browsing habits (e.g. [53]) or spoken phrases in Voice-over-IP connections (e.g. [56]). Therefore TLS may utilize variable-length encoding to conceal the plaintext length up to some granularity.

To capture such requirements, the notion of *stateful length-hiding authenticated encryption* (stateful LHAE) was recently introduced by Paterson et al. [48], as a formalization of the security properties that are expected from the TLS Record Layer. The authors of [48] were able to show that CBC-based Record Layer protocols of TLS 1.1 and 1.2 provably meet this security goal under reasonable assumptions.[3] The results are not applicable to TLS 1.0, since it is well-known that this version is insecure against chosen-plaintext attacks [4, 5] since intialization vectors are not always chosen at random.

DIGITAL SIGNATURES. Our analysis furthermore requires that the employed signature scheme is secure against existential forgeries under adaptive attacks (see Definition 2). The current TLS standards offer three different signature schemes for authentication: RSASSA-PKCS#1 v1.5 [35], DSA [42], and ECDSA [33]. To our knowledge there exists currently no security proof for these signature schemes (under standard complexity assumptions). In the random oracle model, DSA and ECDSA are provably secure. More details can be found in [50, 54].

1.3 Related Work

Because of its eminent role, TLS and its building blocks have been subject to several security analyses. In 1996, Schneier and Wagner presented several minor flaws and some new active attacks against SSL 3.0 [55]. Starting with the famous Bleichenbacher attack [15], many papers focus on various versions of the PKCS#1 standard [35] that defines the encryption padding used in TLS with RSA-encrypted key transport [23, 34, 37, 36]. At Crypto'02, Johnson and Kaliski showed that a simplified version of TLS with padded RSA is

[3] The proceedings version of [48] contains only a proof of *stateless* LHAE security, a full proof was announced for the full version. However, as also noted in [48], it is straightforward to adopt the results to the stateful setting.

IND-CCA secure when modeling TLS as a 'tagged key-encapsulation mechanism' (TKEM) [34] under the strong non-standard assumption that a 'partial RSA decision oracle' is available.

In an independent line of research, several works analysed (simplified versions of) TLS using automated proof techniques in the Dolev-Yao model [28]. Proofs that rely on the Dolev-Yao model view cryptographic operations as deterministic operations on abstract algebras. There has been some work on simplified TLS following the theorem proving and model checking approach, i.e. Mitchell *et al.* used a finite-state enumeration tool named Murphi [44] while Ogata and Futatsugi used the interactive theorem prover OTS/CafeObj [47]. Paulson used the inductive method and the theorem prover Isabelle [49]. Unfortunately it is not known if these proofs are actually cryptographically sound.

Bhargavan *et al.* [13] go two steps farther: First, they automatically derive their formal model from the source code of an TLS implementation, and second they try to automatize computational proofs using the CryptoVerif tool. Chaki and Datta [21] also use source code of TLS, automatically find a weakness in OpenSSL 0.9.6c, and claim that SSL 3.0 is correct.

In 2008, Gajek *et al.* presented the first security analysis of the complete TLS protocol, combining Handshake and Record Layer, in the Universal Composability framework [18] for all three key exchange protocols static Diffie-Hellman, ephemeral signed Diffie-Hellman, and encrypted key transport [32]. The nonces r_C and r_S exchanged between client and server can be seen as an instantiation of the protocol of Barak *et al.* [3] to agree on a globally unique session id. However, the ideal functionalities described in this paper are strictly weaker than the security guarantees we expect from TLS: For the Handshake part, only unauthenticated key exchange is modelled ($\mathcal{F}_{KE}$), and thus the secure communication channel functionality ($\mathcal{F}_{SCS}$) only guarantees confidentiality, not authenticity of endpoints. The paper further assumes that RSA-OEAP is used for encrypted key transport, which is not the case for current versions of TLS.

Küsters and Tuengerthal [40] claim to prove composable security for TLS assuming only local session identifiers, but leave out all details of the proof and only point to [32].

Morissey *et al.* [45] analysed, in a paper that is closest to our results, the security of the truncated TLS Handshake protocol (cf. Section 5) in the random oracle model and provided a modular proof of security for the established application keys. They make extensive use of the random oracle model to separate the three layers they define in the TLS Handshake, and to switch from computational to indistinguishability based security models. The proof of Morissey *et al.* proceeds in three steps, and the order of messages of the TLS Handshake is slightly changed to better separate these three steps. They first consider a very weak class of passively secure key exchange protocols where the session key cannot *be computed* from the session transcript. As an example, when considering encrypted key transport (of the premaster secret) this requirement can easily be fulfilled if the employed public key encryption scheme is OW-CPA secure. Next they define a slightly stronger security notion that additionally protects against unknown key share attacks and show that it applies to the master secret key exchange of TLS. As before security of the key is defined in a one-way sense. In the last step they show that the 'application keys' (i.e. the encryption keys and MAC keys) produced by TLS fulfill the standard notion of security, namely *indistinguishability* from random values. The use of the random oracle model is justified by the authors by the fact that 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 only consider Diffie-Hellman-based ciphersuites.

The work of Morissey *et al.* [45], which can be seen as a reference for the TLS Handshake protocol, considers also security of RSA based ciphersuites, and thus is much broader in scope than our paper, but it does not cover our analysis of the TLS-DHE ciphersuite. The modular proof strategy used in this paper is essentially bound to the Random Oracle Model, since secure protocols for the premaster phase only yield secure protocols for the master phase if the master secret is derived from the premaster secret by evaluating

a Random Oracle. Thus the ROM is used not only to allow a security proof for TLS-RSA ciphersuites, but also to allow for a modular proof technique.

In a very recent work, Paterson, Ristenpart, and Shrimpton [48] introduce the notion of length-hiding authenticated encryption, which aims to capture the properties from the TLS Record Layer protocols. 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. Their paper extends the seminal work of Bellare and Namprempre [8, 9] on authenticated encryption, and on the analysis of different Mac-then-Encode-then-Encrypt (MEE) schemes analysed by Krawczyk [38] and Maurer and Tackmann [43].

This is an important building block for our work, since it allows to capture the precise notion of a TLS-based authenticated and confidential channel establishment protocol (ACCE). TLS-ACCE is used implicitly in many security applications (e.g. the Same Origin Policy of webbrowsers), and explicitely stating the security guarantees offered by TLS-ACCE is an important step towards a future anaysis of these protocols.

Very recently, Brzuska *et al.* [17] 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 secure in a key-indistinguishability-based security model.

2 Preliminaries and Definitions

In this section, we recall the required definitions for our result on the TLS protocol. 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 $a \xleftarrow{\$} A$ 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.

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

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

where $a, b, c \xleftarrow{\$} \mathbb{Z}_q$.

2.2 Digital Signature Schemes

A digital signature scheme is a triple $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$, consisting of a key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$ generating a (public) verification key pk and a secret signing key sk on input of security parameter κ, signing algorithm $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$ generating a signature for message m, and verification algorithm $\mathsf{SIG.Vfy}(pk, \sigma, m)$ returning 1, if σ is a valid signature for m under key pk, and 0 otherwise.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates a public/secret key pair $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$, the adversary receives pk as input.

2. The adversary may query arbitrary messages m_i to the challenger. The challenger replies to each query with a signature $\sigma_i = \mathsf{SIG.Sign}(sk, m_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.

3. Eventually, the adversary outputs a message/signature pair (m, σ).

Definition 2. We say that SIG is *$(t, \epsilon_{\mathsf{SIG}})$-secure* against *existential forgeries under adaptive chosen-message attacks* (EUF-CMA), if for all adversaries $\mathcal{A}$ that run in time t it holds that

$$\Pr\left[(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^{\kappa}, pk) \text{ such that } \mathsf{SIG.Vfy}(pk, m, \sigma) = 1 \wedge m \notin \{m_1, \ldots, m_q\}\right] \leq \epsilon_{\mathsf{SIG}}.$$

Note that we have $q \leq t$, i.e. the number of allowed queries q is bound by the running time t of the adversary.

2.3 Pseudo-Random Functions

A *pseudo-random function* is an algorithm PRF. This algorithm implements a deterministic function $z = \mathsf{PRF}(k, x)$, taking as input a key $k \in \mathcal{K}_{\mathsf{PRF}}$ and some bit string x, 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 challenger samples $k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$ uniformly random.

2. The adversary may query arbitrary values x_i to the challenger. The challenger replies to each query with $z_i = \mathsf{PRF}(k, x_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.

3. Eventually, the adversary outputs value x and a special symbol $\top$. The challenger sets $z_0 = \mathsf{PRF}(k, x)$ and samples $z_1 \xleftarrow{\$} \{0,1\}^{\mu}$ uniformly random. Then it tosses a coin $b \xleftarrow{\$} \{0,1\}$, and returns z_b to the adversary.

4. Finally, the adversary outputs a guess $b' \in \{0,1\}$.

Definition 3. We say that PRF is a *$(t, \epsilon_{\mathsf{PRF}})$-secure* pseudo-random function, if an adversary running in time t has at most an advantage of ϵ_{PRF} to distinguish the PRF from a truly random function, i.e.

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

Again the number of allowed queries q is upper bounded by t (see Def. 2).

Remark 1. In 2008, Fouque *et al.* [31] showed that the HMAC-based key-derivation function of TLS is a pseudo-random function for 1) $\mathcal{K}_{PRF} = S$, where S is a prime-order group of size $|S| = q$ that is either defined over an elliptic curve or as a subgroup of $\mathbb{Z}_p^*$ such that $q|p-1$, and 2) $\mathcal{K}_{PRF} = \{0,1\}^l$ where l is the size of the master-secret ($l = 384$). The underlying security assumptions are all related to the fact that the compression function of the hash function used in HMAC behaves like a pseudo-random function. More details can be found in [31].

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

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

The PRF-Oracle-Diffie-Hellman (PRF-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.

2.4 Stateful Length-Hiding Authenticated Encryption

Let us now describe the stateful variant of LHAE security (the following description and security model was obtained from the authors of [48] via personal communication). See [48] for a detailed discussion and motivation of this model.

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 5. 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.

- 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.

$\underline{\mathsf{Encrypt}(m_0, m_1, \mathsf{len}, H)}$:	$\underline{\mathsf{Decrypt}(C, H)}$:
$u := u + 1$	$v := v + 1$
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_0, st_e)$	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 $C \neq C_v$, then $\mathsf{phase} := 1$
$(C_u, st_e) := (C^{(b)}, st_e^{(b)})$	If $\mathsf{phase} = 1$ then return m
Return C_u	Return $\perp$

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

3 Transport Layer Security

The current version of TLS is 1.2 [27] and coexists with its predecessors TLS 1.0 [25] and TLS 1.1 [26]. 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).

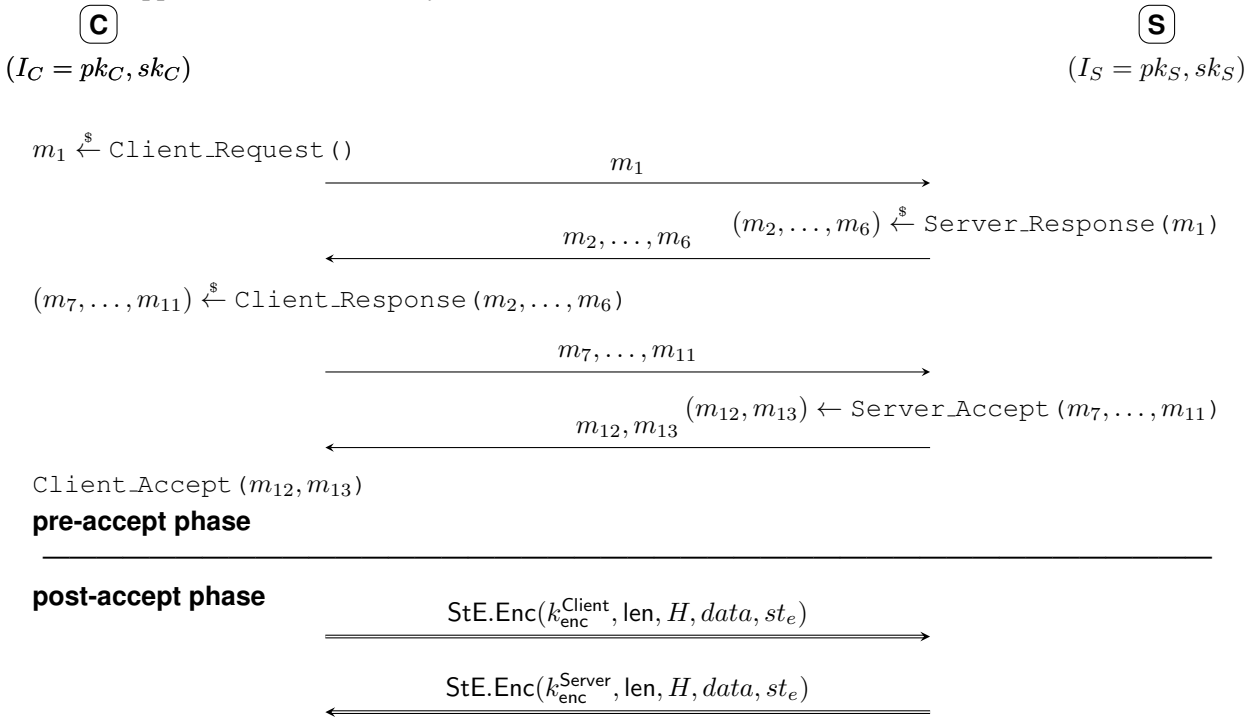

Figure 2: TLS handshake for ciphersuites `TLS_DHE_*` with client authentication

The TLS-DHE 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.

CLIENT HELLO. 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.

SERVER HELLO AND SERVER KEY EXCHANGE. 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. In our analysis the random value r_S is important. The server may send a TLS session ID sID to the client. Message m_3 may contain a chain of certificates, starting from the TLS server certificate up to a direct child of a root certificate. Since we do not include public key infrastructures in our analysis (the identity of each party is its public key pk_S), one certificate $cert_S$ containing pk_S (which may be self-signed) is sufficient for this paper. When the certificate $cert_S$ is received, the client sets its partner id $\Pi := S$. 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'.

CLIENT KEY EXCHANGE AND CLIENT FINISHED. Having received these messages, the signature σ_S is verified. If this fails, the client 'rejects' and aborts. Otherwise, after successful verification, the client is able to complete the key exchange and to compute the cryptographic keys. The `Client Certificate` message m_7 contains a signing certificate $cert_C$ with the public key pk_C of the client. Message m_8 is called `Client Key Exchange`, and contains the Diffie-Hellman share T_C of the client. When the certificate $cert_C$ is received by the server, the server sets its partner id $\Pi := C$. To authenticate the client, a signature σ_C is computed on a concatenation of all previous messages (up to m_8) and padded prefixes, thus including the two random nonces and the two Diffie-Hellman shares. This signature is contained 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 $k_{\mathsf{enc}}^{\mathsf{Client}} := (K_{\mathsf{enc}}^{C\to S}, K_{\mathsf{mac}}^{C\to S})$ and $k_{\mathsf{dec}}^{\mathsf{Client}} := (K_{\mathsf{enc}}^{S\to C}, K_{\mathsf{mac}}^{S\to C})$ is computed as

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

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.

After these computations have been completed, the keys are handed over to the TLS Record Layer of

the client, which is now able to encrypt and MAC 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.

Remark 2. Please note that a padding is applied to fin_C before encryption and that this padding allows for (partially) known plaintext attacks on m_{11}. Thus if we analyse TLS in a key-indistinguishability-based security model, the answer to a `Test` query could be determined by simply decrypting m_{11}, and checking if the resulting plaintext has the appropriate padding. Thus TLS is not provably secure in any such model.

SERVER FINISHED. 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 $k_{\mathsf{enc}}^{\mathsf{Server}} := (K_{\mathsf{enc}}^{S\to C}, K_{\mathsf{mac}}^{S\to C})$ and $k_{\mathsf{dec}}^{\mathsf{Server}} := (K_{\mathsf{enc}}^{C\to S}, K_{\mathsf{mac}}^{C\to S})$ are computed as in (1).[4] 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` 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'.

ENCRYPTED PAYLOAD TRANSMISSION. The obtained keys can now be used to transmit payload data in the TLS Record Layer using a stateful symmetric encryption scheme $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$ (cf. Section 2.4). The CBC-based TLS Record Layer protocols work as follows. The state st_e of the encryption algorithm consists of a sequence number, which is incremented on each encryption operation. The encryption algorithm takes a message m and computes a MAC over m, the sequence counter, and some additional header data H (such as version numbers, for instance). Then message and MAC are encoded into a bit string by using a padding to a specified length len and encrypted ('MAC-then-Encode-then-Encrypt').

The state st_d of the decryption algorithm consists of a sequence number, which is incremented on each decryption operation. Given a ciphertext, the algorithm decrypts and verifies the MAC using its own sequence counter. See [48] for details.

ABBREVIATED TLS HANDSHAKES, SIDE-CHANNELS, AND CROSS-PROTOCOL ATTACKS. In our analysis, we do not consider abbreviated TLS Handshakes, but we note that the server can always enforce a full TLS Handshake. Moreover, we do not consider attacks based on side-channels, such as error messages or implementation issues like the cross-protocol attack from [55].

[4] Note that we have $k_{\mathsf{enc}}^{\mathsf{Server}} = k_{\mathsf{dec}}^{\mathsf{Client}}$ and $k_{\mathsf{dec}}^{\mathsf{Server}} = k_{\mathsf{enc}}^{\mathsf{Client}}$.

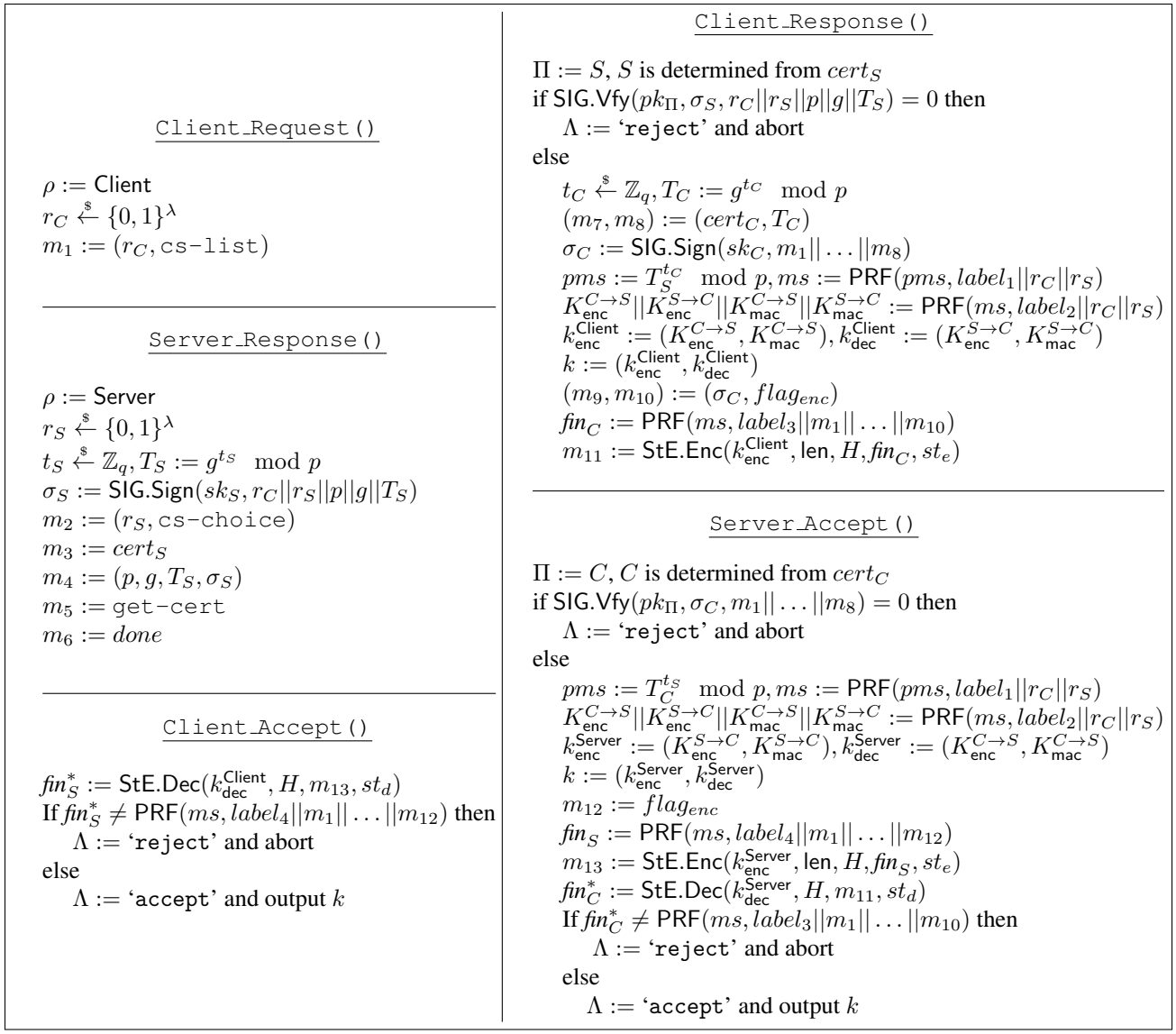

Figure 3: Computation of Client/Server Handshake Messages

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 [14, 19, 41, 24] dates back to Bellare and Rogaway [11], where an adversary is provided with an 'execution environment', which emulates the real-world capabilities of an active adversary. In this model, the adversary has full control over the communication network, which allows him to forward, alter, or drop any message sent by the participants, or insert new messages. 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.

4.1 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:

- $\Lambda \in \{\texttt{accept}, \texttt{reject}\}$.
- $k \in \mathcal{K}$, 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 within the protocol execution.[5]
- 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 the transcript of all messages sent/received during the TLS Handshake).

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

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

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.[6]

  The variables Λ, k, Π, st are also set after a Send-query. When and how depends on the considered protocol.

- $\mathsf{Reveal}(\pi_i^s)$: 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 $\Lambda = \text{`}\texttt{accept}'$, see (2).

[5] 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.

[6] Note that we do not include the identity of the (intended) communication partner in the Send-query. Instead, we assume that the exchange of identities of communication partners (which is necessary to determine the public-key used to perform authentication) is part of the protocol.

- $\mathsf{Corrupt}(P_i)$: Oracle π_i^1 responds with the long-term secret key sk_i of party P_i.[7] 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 $\perp$. 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.

The Send-query enables the adversary to initiate and run an arbitrary number of protocol instances, sequential or in parallel, and provides full control over the communication between all parties. The Reveal-query may be used to learn the session keys used in previous/concurrent protocol executions. The Corrupt-query allows the attacker to learn sk_i of party P_i, it may for instance be used by $\mathcal{A}$ to impersonate P_i. The Test-query will be used to define security.

4.2 Security Definition

Bellare and Rogaway [11] 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 6 (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).

Remark 3. We remark that matching conversations in the above sense can also be seen as *post-specified session identifiers*. The 'asymmetry' of the definition (i.e., the fact that we have to distinguish which party has sent the last message) is necessary, due to the fact that protocol messages are sent sequentially. For instance in the TLS Handshake protocol (see Figure 2) the last message of the client is the 'client finished' message fin_C, and then it waits for the 'server finished' message fin_S before acceptance. In contrast, the server sends fin_S *after* receiving fin_C. Therefore the server has to 'accept' without knowing whether its last message was received by the client correctly. We have to take this into account in the definition of matching conversations, since it will later be used to define security of the protocol in presence of an active adversary that simply drops the last protocol message.

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, thus an adversary cannot distinguish the session key k from a random key.

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

[7] 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.

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 7. 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 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 intended 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.

   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

   - 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 $\tau_0 < \tau_j$,
   - $\mathcal{A}$ 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

   then the probability that $\mathcal{A}$ outputs b' which equals the bit b sampled by the Test-query satisfies

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

   If an adversary $\mathcal{A}$ outputs b' such that $b' = b$ and the above conditions are met, then we say that $\mathcal{A}$ *anwers the* Test*-challenge correctly*.

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

Remark 4. 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.

5 Truncated TLS with Ephemeral Diffie-Hellman is a Secure AKE Protocol

In this section we 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 `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 [45, 46]. 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

[8] That is, P_j is not corrupted when π_i^s 'accepts'. Recall that uncorrupted parties are τ-corrupted with $\tau = \infty$.

- message m_{11} contains only fin_C (instead of $\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, \mathit{fin}_C, st_e)$), and
- message m_{13} contains only fin_S (instead of $\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, \mathit{fin}_S, st_e)$).

This simple modification allows to prove security in the key-indisinguishability-based security model from Section 4.

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* PRF-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 ephemeral Diffie-Hellman TLS Handshake protocol in the sense of Definition 7 with $t \approx t'$ holds that

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

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 proving 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

$$\epsilon_{\mathsf{ttls}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + \epsilon_{\mathsf{ke}}.$$

5.1 Authentication

Lemma 1. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s with $\rho = \mathsf{Client}$ 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. The proof proceeds in a *sequence of games*, following [12, 52]. 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 $\mathsf{break}_\delta^{(1)}$ be the event that occurs when the first oracle that accepts maliciously in the sense of Definition 7 with $\rho = \mathsf{Client}$ in Game δ.

Game 0. This game equals the AKE security experiment described in Section 4. Thus, for some $\epsilon_{\mathsf{client}}$ we have

$$\Pr[\mathsf{break}_0^{(1)}] = \epsilon_{\mathsf{client}}.$$

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_C or r_S which is not unique. More precisely, the game is aborted if the adversary ever makes a first Send query to an oracle π_i^s, and the oracle replies with random nonce r_C or r_S such that there exists some other oracle $\pi_{i'}^{s'}$ which has previously sampled the same nonce.

In total less than $d\ell$ nonces r_C and r_S are sampled, each uniformly random from $\{0,1\}^\lambda$. Thus, the probability that a collision occurs is bounded by $(d\ell)^2 2^{-\lambda}$, which implies

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

Note that now each oracle has a unique nonce r_C or r_S, 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 client oracle will be the first oracle to accept maliciously. If our guess is wrong, i.e. if there is another (Client or Server) oracle that accepts before, then we abort the game.

Technically, this game is identical, except for the following. The challenger guesses two random indices $(i^*, s^*) \xleftarrow{\$} [\ell] \times [d]$. If there exists an oracle π_i^s that 'accepts' maliciously, and $(i,j) \neq (i^*, j^*)$ and π_i^s has $\rho \neq \mathsf{Client}$, then the challenger aborts the game. Note that if the first oracle π_i^s that 'accepts' maliciously has $\rho = \mathsf{Client}$, then with probability $1/(d\ell)$ we have $(i,j) = (i^*, j^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = d\ell \cdot \Pr[\mathsf{break}_2^{(2)}].$$

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 and has $\rho = \mathsf{Client}$, as otherwise the game is aborted.

Game 3. Again the challenger proceeds as before, but we add an abort rule. We want to make sure that $\pi_{i^*}^{s^*}$ receives as input exactly the Diffie-Hellman value T_S that was selected by some other uncorrupted oracle that received the nonce r_C chosen by $\pi_{i^*}^{s^*}$ as first input (note that there may be several such oracles, since the attacker may send copies of r_C to many oracles).

Technically, we abort and raise event $\mathsf{abort}_{\mathsf{sig}}$, if oracle $\pi_{i^*}^{s^*}$ ever receives as input a message $m_3 = cert_S$ indicating intended partner $\Pi = j$ and message $m_4 = (p, g, T_S, \sigma_S)$ such that σ_S is a valid signature over $r_C||r_S||p||g||T_S$, but there exists no oracle π_j^t which has previously output σ_S. Clearly we have

$$Pr[\mathsf{break}_2^{(1)}] \leq \Pr[\mathsf{break}_3^{(1)}] + \Pr[\mathsf{abort}_{\mathsf{sig}}].$$

Note that the experiment is aborted, if $\pi_{i^*}^{s^*}$ does not accept *maliciously*, due to Game 2. This means that party P_j must be τ_j-corrupted with $\tau_j = \infty$ (i.e., not corrupted) when $\pi_{i^*}^{s^*}$ accepts (as otherwise $\pi_{i^*}^{s^*}$ does not accept *maliciously*). To show that $\Pr[\mathsf{abort}_{\mathsf{sig}}] \leq \ell \cdot \epsilon_{\mathsf{sig}}$, we construct a signature forger as follows. The forger receives as input a public key pk^* and simulates the challenger for $\mathcal{A}$. It guesses an index $\phi \xleftarrow{\$} [\ell]$, sets $pk_\phi = pk^*$, and generates all long-term public/secret keys as before. Then it proceeds as the challenger in Game 3, except that it uses its chosen-message oracle to generate a signature under pk_ϕ when necessary.

If $\phi = j$, which happens with probability $1/\ell$, then the forger can use the signature received by $\pi_{i*}^{s^*}$ to break the EUF-CMA security of the signature scheme with success probability ϵ_{sig}, so $\Pr[\mathsf{abort}_{\mathsf{sig}}]/\ell \leq \epsilon_{\mathsf{sig}}$. Therefore if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

$$Pr[\mathsf{break}_2^{(1)}] \leq \Pr[\mathsf{break}_3^{(1)}] + \ell \cdot \epsilon_{\mathsf{sig}}.$$

Note that in Game 3 oracle $\pi_{i*}^{s^*}$ receives as input a Diffie-Hellman value T_S such that T_S was chosen by another oracle, but not by the attacker. Note also that there may be multiple oracles that issued a signature σ_S containing r_C, since the attacker may have sent several copies of r_C to several oracles.

Game 4. In this game we want to make sure that we know *which* oracle π_j^t will issue the signature σ_S that $\pi_{i*}^{s^*}$ receives. Note that this signature includes the random nonce r_S, which is unique due to Game 1. Therefore the challanger in this game proceeds as before, but additionally guesses two indices $(j^*, t^*) \xleftarrow{\$} [\ell] \times [d]$. It aborts, if the attacker does *not* make a Send-query containing r_C to $\pi_{j*}^{t^*}$, and $\pi_{j*}^{t^*}$ responds with messages containing σ_S such that σ_S is forwarded to $\pi_{i*}^{s^*}$.

We know that there must exists at least one oracle that outputs σ_S such that σ_S is forwarded to $\pi_{i*}^{s^*}$, due to Game 3. Thus we have

$$\Pr[\mathsf{break}_3^{(1)}] \leq d\ell \cdot \Pr[\mathsf{break}_4^{(1)}].$$

Note that in this game we know exactly that oracle $\pi_{j*}^{t^*}$ chooses the Diffie-Hellman share T_S that $\pi_{i*}^{s^*}$ uses to compute its premaster secret.

Game 5. Recall that $\pi_{i*}^{s^*}$ computes the master secret as $ms = \mathsf{PRF}(T_S^{t_c}, label_1||r_C||r_S)$, where T_S denotes the Diffie-Hellman share received from $\pi_{j*}^{t^*}$, and t_c denotes the Diffie-Hellman exponent chosen by $\pi_{i*}^{s^*}$. In this game we replace the master secret ms computed by $\pi_{i*}^{s^*}$ with an independent random value $\widetilde{ms}$. Moreover, if $\pi_{j*}^{t^*}$ receives as input the same Diffie-Hellman share T_C that was sent from $\pi_{i*}^{s^*}$, then we set the master secret of $\pi_{j*}^{t^*}$ equal to $\widetilde{ms}$. Otherwise we compute the master secret as specified in the protocol. We claim that

$$Pr[\mathsf{break}_4^{(1)}] \leq \Pr[\mathsf{break}_5^{(1)}] + \epsilon_{\mathsf{PRF\text{-}ODH}}.$$

Suppose there exists an adversary $\mathcal{A}$ that distinguishes Game 5 from Game 4. We show that this implies an adversary $\mathcal{B}$ that solves the PRF-ODH problem.

Adversary $\mathcal{B}$ outputs $(label_1||r_C||r_S)$ to its oracle and receives in response (g, g^u, g^v, R), where either $R = \mathsf{PRF}(g^{uv}, label_1||r_C||r_S)$ or $R \xleftarrow{\$} \{0,1\}^\mu$. It runs $\mathcal{A}$ by implementing the challenger for $\mathcal{A}$, and embeds (g^u, g^v) as follows. Instead of letting $\pi_{i*}^{s^*}$ choose $T_C = g^{t_C}$ for random $t_C \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{B}$ defines $T_C := g^u$. Similarly, the Diffie-Hellman share T_S of $\pi_{j*}^{t^*}$ is defined as $T_S := g^v$. Finally, the master secret of $\pi_{i*}^{s^*}$ is set equal to R.

Note that $\pi_{i*}^{s^*}$ computes the master secret after receiving T_S from $\pi_{j*}^{t^*}$, and then it sends $m_8 = T_C$. If the attacker decides to forward m_8 to $\pi_{j*}^{t^*}$, then the master secret of $\pi_{j*}^{t^*}$ is set equal to R. If $\pi_{j*}^{t^*}$ receives $T_{C'} \neq T_C$, then $\mathcal{B}$ queries its oracle to compute $ms' = \mathsf{PRF}(T_{C'}^v, label_1||r_C||r_S)$, and sets the master secret of $\pi_{j*}^{t^*}$ equal to ms'.

Note that in any case algorithm $\mathcal{B}$ 'knows' the master secret of $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$, and thus is able to compute all further protocol messages (in particular the finished messages fin_C and fin_S) and answer a potential Reveal-query to $\pi_{j*}^{t^*}$ as required (note that there is no Reveal-query to $\pi_{i*}^{s^*}$, as otherwise the experiment is aborted, due to Game 2). If $R = \mathsf{PRF}(g^{uv}, label_1||r_C||r_S)$, then the view of $\mathcal{A}$ is identical to Game 4, while if $R \xleftarrow{\$} \{0,1\}^\mu$ then it is identical to Game 5, which yields the above claim.

Game 6. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i*}^{s^*}$ with a random function. If $\pi_{j*}^{t^*}$ uses the same master secret $\widetilde{ms}$ as $\pi_{i*}^{s^*}$ (cf. Game 5), then the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{j*}^{t^*}$ is replaced as well. Of course the same random function is used for both oracles sharing the same $\widetilde{ms}$. In particular, this function is used to compute the `Finished` messages by both partner oracles.

Distinguishing Game 6 from Game 5 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\Pr[\mathsf{break}_5^{(1)}] \leq \Pr[\mathsf{break}_6^{(1)}] + \epsilon_{\mathsf{prf}}$$

Game 7. 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 6. 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 $\pi_{j*}^{t^*}$.

Thus, this game proceeds exactly like the previous game, except that the challenger now aborts if oracle $\pi_{i*}^{s^*}$ accepts without having a matching conversation to $\pi_{j*}^{t^*}$. Thus we have $\Pr[\mathsf{break}_7^{(1)}] = 0$.

The `Finished` messages are computed by evaluating a truly random function $F_{\widetilde{ms}}$, 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 $F_{\widetilde{ms}}(label_3||m_1||\cdots||m_{12})$. Therefore we have $\Pr[\mathsf{break}_7^{(1)}] = 2^{-\mu}$ and

$$\Pr[\mathsf{break}_6^{(1)}] \leq \Pr[\mathsf{break}_7^{(1)}] + \frac{1}{2^\mu} = \frac{1}{2^\mu}.$$

Collecting probabilities from Game 0 to Game 7 yields Lemma 1. □

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*

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

where all quantities are defined as stated in Theorem 1.

PROOF. Let $\mathsf{break}_\delta^{(2)}$ be the event that occurs when the first oracle that accepts maliciously in the sense of Definition 7 with $\rho = \mathsf{Server}$ in Game δ.

Game 0. This game equals the AKE security experiment described in Section 4. Thus, for some $\epsilon_{\mathsf{server}}$ we have

$$\Pr[\mathsf{break}_0^{(2)}] = \epsilon_{\mathsf{server}}.$$

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_C or r_S which is not unique. With the same arguments as in Game 1 from the proof of Lemma 1 we have

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

Game 2. This game is identical, except for the following. The challenger guesses two random indices $(i^*, s^*) \xleftarrow{\$} [\ell] \times [d]$. If there exists an oracle π_i^s that 'accepts' maliciously, and $(i, j) \neq (i^*, j^*)$ and π_i^s has $\rho \neq \mathsf{Server}$, then the challenger aborts the game. Note that if the first oracle π_i^s that 'accepts' maliciously has $\rho = \mathsf{Server}$, then with probability $1/(d\ell)$ we have $(i, j) = (i^*, j^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = d\ell \cdot \Pr[\mathsf{break}_2^{(2)}].$$

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 and has $\rho = \mathsf{Server}$, as otherwise the game is aborted.

Game 3. The challenger proceeds as before, but we add an abort rule. We want to make sure that $\pi_{i*}^{s^*}$ receives as input exactly the Diffie-Hellman value $m_8 = T_C$ that was selected by some other uncorrupted oracle.

Technically, we abort and raise event $\mathsf{abort}_{\mathsf{sig}}$, if oracle $\pi_{i*}^{s^*}$ ever receives as input a message $m_7 = cert_C$ indicating intended partner $\Pi = j$ and message $m_9 = \sigma_C = \mathsf{SIG.Sign}(sk_C, m_1 || \ldots, ||m_8)$ such that σ_C is a valid signature but there exists no oracle π_j^t which has previously output σ_C. Clearly we have

$$Pr[\mathsf{break}_2^{(2)}] \leq \Pr[\mathsf{break}_3^{(2)}] + \Pr[\mathsf{abort}_{\mathsf{sig}}].$$

Note that the experiment is aborted, if $\pi_{i*}^{s^*}$ does not accept *maliciously*, due to Game 2. This means that party P_j must be τ_j-corrupted with $\tau_j = \infty$ (i.e., not corrupted) when $\pi_{i*}^{s^*}$ accepts. To show that $\Pr[\mathsf{abort}_{\mathsf{sig}}] \leq \ell \cdot \epsilon_{\mathsf{sig}}$, we construct a signature forger as follows. The forger receives as input a public key pk^* and simulates the challenger for $\mathcal{A}$. It guesses an index $\phi \xleftarrow{\$} [\ell]$, sets $pk_\phi = pk^*$, and generates all long-term public/secret keys as before. Then it proceeds as the challenger in Game 3, except that it uses its chosen-message oracle to generate a signature under pk_ϕ when necessary.

If $\phi = j$, which happens with probability $1/\ell$, then the forger can use the signature received by $\pi_{i*}^{s^*}$ to break the EUF-CMA security of the signature scheme with success probability ϵ_{sig}, so $\Pr[\mathsf{abort}_{\mathsf{sig}}]/\ell \leq \epsilon_{\mathsf{sig}}$. Therefore if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

$$Pr[\mathsf{break}_2^{(2)}] \leq \Pr[\mathsf{break}_3^{(2)}] + \ell \cdot \epsilon_{\mathsf{sig}}.$$

Note that in Game 3 oracle $\pi_{i*}^{s^*}$ receives as input a Diffie-Hellman value T_C such that T_C was chosen by another oracle, but not by the attacker. Note also that this oracle is unique, since the signature includes the client nonce r_C, which is unique due to Game 1. From now on we denote this unique oracle with $\pi_{j*}^{t^*}$.

Note also that $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$ share a premaster secret $pms = T_C^{t_S} = T_S^{t_C}$, where $T_C = g^{t_C}$ and $T_S = g^{t_S}$ for random exponents t_S and t_C chosen by $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$, respectively.

Game 4. In this game, we replace the premaster secret $pms = g^{t_C t_S}$ shared by $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$ with a random value g^r, $r \xleftarrow{\$} \mathbb{Z}_q$. The fact that the challenger has full control over the Diffie-Hellman shares T_C and T_S exchanged between $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$, due to the modifications introduced in the previous games, provides us with the leverage to prove indistinguishability under the Decisional Diffie-Hellman assumption.

Technically, the challenger in Game 4 proceeds as before, but when $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$ compute the premaster secret as $pms = g^{t_C t_S}$, the challenger replaces this value with a uniformly random value $\widetilde{pms} = g^r, r \xleftarrow{\$} \mathbb{Z}_p^*$, which is in the following used by both partner oracles.

Suppose there exists an algorithm distinguishing Game 4 from Game 3. Then we can construct an algorithm $\mathcal{B}$ solving the DDH problem as follows. Algorithm $\mathcal{B}$ receives as input a DDH challenge (g, g^u, g^v, g^w). The challenger defines $T_C := g^u$ and $T_S := g^v$ for the Diffie-Hellman shares chosen by $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$, respectively. Instead of computing the Diffie-Hellman key as in Game 3, it sets $pms = g^w$ both for the 'client' and the 'server' oracle. Now if $w = uv$, then this game proceeds exactly like Game 3, while if w is random than this game proceeds exactly like Game 4. The DDH assumption therefore implies that

$$\Pr[\mathsf{break}_3^{(2)}] \leq \Pr[\mathsf{break}_4^{(2)}] + \epsilon_{\mathsf{ddh}}.$$

Note that in Game 4 the premaster secret of $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$ is uniformly random, and independent of T_C and T_S. This will provide us with the leverage to replace the function $\mathsf{PRF}(\widetilde{pms}, \cdot)$ with a truly random function in the next game.

Game 5. In Game 5 we make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$ is chosen uniformly random, and independent of T_C and T_S. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1||r_C||r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 5 from Game 4 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\Pr[\mathsf{break}_4^{(2)}] \leq \Pr[\mathsf{break}_5^{(2)}] + \epsilon_{\mathsf{prf}}.$$

Game 6. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i*}^{s^*}$ and $\pi_{j*}^{t^*}$ with a random function. 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 `Finished` messages by both partner oracles.

Distinguishing Game 6 from Game 5 again implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\Pr[\mathsf{break}_5^{(2)}] \leq \Pr[\mathsf{break}_6^{(2)}] + \epsilon_{\mathsf{prf}}.$$

Game 7. 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 $\pi_{j*}^{t^*}$ due to Game 6. 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 $\pi_{j*}^{t^*}$.

Thus, this game proceeds exactly like the previous game, except that the challenger now aborts if oracle $\pi_{i*}^{s^*}$ accepts without having a matching conversation to $\pi_{j*}^{t^*}$. Therefore we have $\Pr[\mathsf{break}_7^{(1)}] = 0$.

The `Finished` messages are computed by evaluating a truly random function $F_{\widetilde{ms}}$, 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 $F_{\widetilde{ms}}(label_3||m_1||\cdots||m_{10})$. Thus we have

$$\Pr[\mathsf{break}_6^{(1)}] \leq \Pr[\mathsf{break}_7^{(1)}] + \frac{1}{2^\mu} = \frac{1}{2^\mu}.$$

Collecting probabilities from Game 0 to Game 7 yields Lemma 2. □

5.2 Indistinguishability of Keys

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 7 (cf. Lemmas 1 and 2) and all other quantities are defined as stated in Theorem 1.

PROOF. Assume without loss of generality that the $\mathcal{A}$ always asks a Test-query such that all conditions in Property 2 of Definition 7 are satisfied. Let $\mathsf{break}_{\delta}^{(3)}$ denote the event that $b' = b$ in Game δ, where b is the random bit sampled by the Test-query, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_{\delta} := \Pr[\mathsf{break}_{\delta}^{(3)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

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

$$\Pr[\mathsf{break}_0^{(3)}] = \frac{1}{2} + \epsilon_{\mathsf{ke}} = \frac{1}{2} + \mathsf{Adv}_0.$$

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 7. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}},$$

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 7 (cf. Lemmas 1 and 2).

Recall that we assume that $\mathcal{A}$ always asks a Test-query such that all conditions in Property 2 of Definition 7 are satisfied. In particular it asks a Test-query to an oracle π_i^s that 'accepts' after the τ_0-th query of $\mathcal{A}$ with intended partner $\Pi = j$, such that 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 indices $(i^*, s^*) \xleftarrow{\$} [\ell] \times [d]$. It aborts and chooses b' at random, if the attacker issues a $\mathsf{Test}(\pi_i^s)$-query with $(i, j) \neq (i^*, s^*)$. With probability $1/(d\ell)$ we have $(i, j) = (i^*, s^*)$, and thus

$$\mathsf{Adv}_1 \leq d\ell \cdot \mathsf{Adv}_2.$$

Note that in Game 2 we know that $\mathcal{A}$ will issue a Test-query to oracle $\pi_{i^*}^{s^*}$. Note also that $\pi_{i^*}^{s^*}$ has a unique 'partner' due to Game 1. In the sequel we denote with $\pi_{j^*}^{t^*}$ the unique oracle such that $\pi_{i^*}^{s^*}$ has a matching conversation to $\pi_{j^*}^{t^*}$, and say that $\pi_{j^*}^{t^*}$ is the *partner* of $\pi_{i^*}^{s^*}$.

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 premaster secret as $pms = g^{uv}$.

The challenger in this game proceeds as before, but replaces the premaster secret pms of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random group element $\widetilde{pms} = g^w$, $w \xleftarrow{\$} \mathbb{Z}_q$. Note that both g^u and g^v are chosen by oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively, as otherwise $\pi_{i^*}^{s^*}$ would not have a matching conversation to $\pi_{j^*}^{t^*}$ and the game would be aborted.

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^w). 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 premaster secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ equal to $pms := g^w$. 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 $pms = g^w$. Since all other oracles are not modified, $\mathcal{B}$ can simulate these oracles properly as well.

If $w = uv$, then the view of $\mathcal{A}$ when interacting with $\mathcal{B}$ is identical to Game 2, while if $w \xleftarrow{\$} \mathbb{Z}_q$ 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. In Game 4 we make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly random. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1||r_C||r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 4 from Game 3 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{prf}}.$$

Game 5. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function $F_{\widetilde{ms}}$. 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

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

Distinguishing Game 5 from Game 4 again implies an algorithm breaking the security of the pseudo-random function PRF. Moreover, in Game 5 the adversary always receives a random key in response to a Test-query, thus receives no information about b', which implies $\mathsf{Adv}_5 = 0$ and

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + \epsilon_{\mathsf{prf}} = \epsilon_{\mathsf{prf}}.$$

Collecting probabilities from Game 0 to Game 5 yields Lemma 3. □

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

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

which yields Theorem 1.

6 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 7). This phase ends, when both communication partners reach an `accept`-state (i.e. $\Lambda =$ '`accept`').

Post-accept phase. This phase is entered, when both communication partners reach an `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 [48]. Technically, we provide a slightly modified execution environment that extends the types of queries an adversary may issue.

6.1 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 Encrypt-query,[9] and chosen-ciphertexts by making a Decrypt-query. Moreover, each oracle π_i^s keeps as additional internal state a bit $b_i^s \xleftarrow{\$} \{0,1\}$, chosen at random at the beginning of the game, 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 $k = (k^\rho_{\mathsf{enc}}, k^\rho_{\mathsf{dec}})$ 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 may interact with the provided oracles by issuing the following queries.

- $\mathsf{Send}^{\mathsf{pre}}(\pi_i^s, m)$: 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 $\Lambda =$ '`accept`'. (Send-queries in an accept-state are handled by the 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 $\Lambda \neq$ '`accept`' then π_i^s returns $\perp$. Otherwise, it proceeds as depicted 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.
- $\mathsf{Decrypt}(\pi_i^s, C, H)$: This query takes as input a ciphertext C and header data H. If π_i^s has $\Lambda \neq$ '`accept`' then π_i^s returns $\perp$. Otherwise, it proceeds as depicted in Figure 4.

[9] This models that an adversary may trick one party into sending some adversarially chosen data. A practical example for this attack scenario are cross-site request forgeries [57] on web servers, or Bard's chosen-plaintext attacks on SSL3.0 [4, 5].

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

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

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

6.2 Security Definition

Security of ACCE protocols is defined by requiring that (i) the protocol is a secure authentication protocol and (ii) in the post-accept phase all data is transmitted over an authenticated and confidential channel in the sense of Definition 5.

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 8. 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 $\tau_0 < \tau_j$, 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 accept 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' 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

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

   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 5. 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 an `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 an `accept`-state, which provides security against *key-compromise impersonation* attacks.

6.3 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.

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 7, and the 'symmetric part' consists of a suitable authenticated symmetric encryption scheme (e.g. secure according to Definition 5).

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 7, 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.

7 TLS with Ephemeral Diffie-Hellman is a Secure ACCE Protocol

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 ephemeral Diffie-Hellman TLS protocol in the sense of Definition 8 with $t \approx t'$ holds that

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

To prove Theorem 2, we divide 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 two lemmas. Lemma 4 bounds the probability $\epsilon_{\mathsf{client}}$ that an authentication-adversary succeeds, Lemma 5 bounds the probability ϵ_{enc} that an encryption-adversary succeeds. Then we have

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

We prove Theorem 2 via the following lemmas.

Lemma 4. *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*

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

where all quantities are defined as stated in Theorem 2.

Note that $\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$, where $\epsilon_{\mathsf{client}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Client}$ that accepts maliciously in the sense of Definition 8, and $\epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Client}$ that accepts maliciously. We claim that

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

and thus

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

The bounds on $\epsilon_{\mathsf{client}}$ and $\epsilon_{\mathsf{server}}$ are derived exactly like in the proofs of Lemma 1 and Lemma 2, therefore we omit the details.

Lemma 5. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ anwers the encryption-challenge correctly is at most $1/2 + \epsilon$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + d\ell \left(\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} \right).$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 8 (cf. Lemma 4) and all other quantities are defined as stated in Theorem 2.

The proof of this lemma extends the proof of Lemma 3 by one game-hop that exploits the sLHAE-security of the encryption scheme.

PROOF. Assume without loss of generality that the $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 8 are satisfied. Let $\mathsf{break}_\delta^{(4)}$ denote the event that $b' = b$ in Game δ, where b is the random bit sampled by the Test-query, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(4)}] - 1/2$ 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

$$\Pr[\mathsf{break}_0^{(3)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0.$$

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 8. 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 8 (cf. Lemma 4).

Recall that we assume that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 8 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 $\Pi = j$, 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 indices $(i^*, s^*) \xleftarrow{\$} [\ell] \times [d]$. It aborts and chooses b' at random, if the attacker outputs (i, s, b') with $(i, j) \neq (i^*, s^*)$. With probability $1/(d\ell)$ we have $(i, j) = (i^*, s^*)$, and thus

$$\mathsf{Adv}_1 \leq d\ell \cdot \mathsf{Adv}_2.$$

Note that in Game 2 we know that $\mathcal{A}$ will output (i^*, s^*, b'). Note also that $\pi_{i^*}^{s^*}$ has a unique 'partner' due to Game 1. In the sequel we denote with $\pi_{j^*}^{t^*}$ the unique oracle such that $\pi_{i^*}^{s^*}$ has a matching conversation to $\pi_{j^*}^{t^*}$, and say that $\pi_{j^*}^{t^*}$ is the *partner* of $\pi_{i^*}^{s^*}$.

Game 3. The challenger in this game proceeds as before, but replaces the premaster secret pms of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random group element $\widetilde{pms} = g^w$, $w \xleftarrow{\$} \mathbb{Z}_q$. Note that both g^u and g^v are chosen by oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively, as otherwise $\pi_{i^*}^{s^*}$ would not have a matching conversation to $\pi_{j^*}^{t^*}$ and the game would be aborted.

With the same arguments as in Game 3 in the proof of Lemma 3, we have

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{ddh}}.$$

Game 4. As in Game 4 in the proof of Lemma 3, we now make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly random. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1||r_C||r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 4 from Game 3 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{prf}}.$$

Game 5. As in Game 5 in the proof of Lemma 3, we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function $F_{\widetilde{ms}}$. 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

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

Distinguishing Game 5 from Game 4 again implies an algorithm breaking the security of the pseudo-random function PRF, 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 uniformly at 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 (Definition 5). 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 5.

Observe that the values generated in this game are exactly distributed as in the previous game. We thus have

$$\mathsf{Adv}_5 = \mathsf{Adv}_6.$$

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 $\epsilon_{\mathsf{sLHAE}}$ in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_6 \leq 1/2 + \epsilon_{\mathsf{sLHAE}}.$$

□

Addig up probabilities from Lemmas 4 and 5, we obtain that

$$\begin{aligned}
\epsilon_{\mathsf{tls}} &\leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}} \\
&\leq 2 \cdot \epsilon_{\mathsf{auth}} + d\ell \left(\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}\right) \\
&\leq 4 \cdot d\ell \left(\frac{d\ell}{2^\lambda} + \ell \cdot \epsilon_{\mathsf{sig}} + \frac{5}{4} \cdot \epsilon_{\mathsf{ddh}} + \frac{5}{2} \cdot \epsilon_{\mathsf{prf}} + \frac{1}{4} \cdot \epsilon_{\mathsf{sLHAE}} + d\ell \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^\mu}\right)\right)
\end{aligned}$$

which yields Theorem 2.

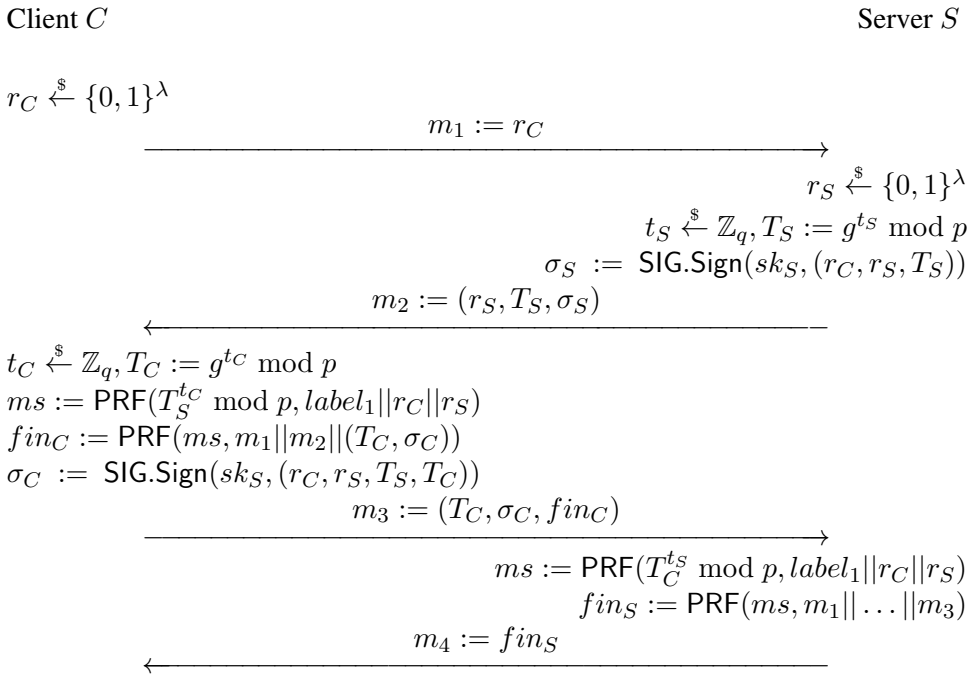

Figure 5: Abstraction of the TLS-DHE handshake.

8 On Proving Security of TLS-DHE from Standard Assumptions

In this section we illustrate why we had to make the PRF-ODH-assumption in the proofs of Lemmas 1 and 4, and why we can prove Lemma 2 based on the standard DDH assumption. In order to allow a comprehensive exposition, let us consider the simplified protocol from Figure 5 which abstracts the TLS-DHE Handshake.

Let us start with an informal description of the issue. Suppose we are given a Client-adversary, that is, an adversary which always makes Client-oracle $C := \pi_i^s$ (i.e., a particular oracle 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), and that we want to assume only that the PRF is secure in the standard sense (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 replace pms with a random value and 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, 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 between C and S.

2. The C will respond with $m_3 = (T_C, \sigma_C, fin_C)$. 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) $ms^* := \mathsf{PRF}(T_S^{t^*}, label_1||r_C||r_S)$ using knowledge of t^*, and

   (d) $\mathit{fin}_C^* := \mathsf{PRF}(ms^*, m_1||m_2||(T^*, \sigma^*))$.

   and sends $m_3^* := (T^*, \sigma^*, \mathit{fin}_C^*)$ 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 all keys, in particular $pms^* := T_S^{t^*}$, 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.

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 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 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

$$\mathit{fin}_S^* = \mathsf{PRF}(ms, m_1|| \ldots ||m_3) \quad \text{and} \quad ms = \mathsf{PRF}(T_S^{t^*}, label_1||r_C||r_S),$$

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 PRF-ODH-assumption, we can however use the given oracle to compute first ms, and from this the `Finished`-message fin_S^*.

SERVER-ADVERSARIES. Interestingly, the above technical problem does not appear if we consider Server-adversaries (i.e., adversaries that make an oracle π_i^s accept maliciously with $\rho = \mathsf{Server}$) instead of Client-adversaries. This is due to the asymmetry of the TLS-DHE Handshake protocol. The reason is, that in this case the adversary is not allowed to corrupt the intended partner of the server (in order to exclude trivial attacks), and is therefore not able to inject an adversarially-chosen Diffie-Hellman share T^*. Note here that the signature sent from the client to the server is computed over *both* Diffie-Hellman shares received and chosen by the client. Therefore in this case the server is able to verify whether its intended partner has received the correct Diffie-Hellman share, and thus the standard DDH assumption is sufficient to prove security.

DISALLOWING CORRUPTIONS. One possibility to circumvent the above problem, and thus to avoid the PRF-ODH-assumption, is to consider a weaker security model. If we disallow Corrupt-queries in the model, then the adversary will not be able to inject an adversarially-chosen, validly signed Diffie-Hellman share. This prevents the above 'test' and again allows a proof under the DDH assumption. However, a security model without corruptions is rather weak. Albeit it may be reasonable for certain applications, it is certainly not adequate for the way how TLS-DHE is used on the Internet.

ADOPTING Σ_0 TO TLS. In [20] Canetti and Krawczyk describe a protocol called Σ_0, which exhibits many similarities to the TLS-DHE Handshake protocol and the simplified protocol from Figure 5, but is provably

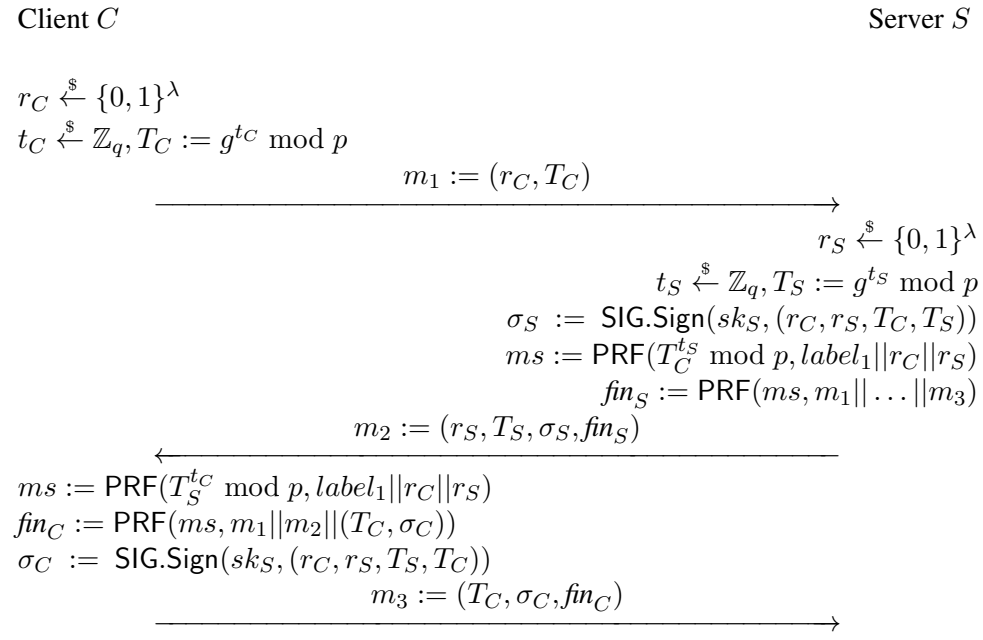

Figure 6: Provably secure TLS-DHE Adopting Σ_0 variant, adopted to the client/server setting and our notation.

secure under standard assumptions (in particular under DDH instead of PRF-ODH). Let us discuss why the differences between Σ_0 and TLS-DHE, albeit subtle, are crucial.

In Figure 6 we describe a simple variant Σ of Σ_0, which essentially extends Σ_0 with a server nonce r_S and replaces the MAC computed over identities in Σ_0 with a MAC (implemented with PRF) computed over all previous protocol messages. Note that these messages include the identities, so the security analysis of Σ_0 carries over to Σ.

The major difference between Σ_0 and TLS-DHE is, that the client 'accepts' already after receiving m_2. There is no message m_4 sent from the server to the client (which thus need not be simulated in a security experiment). This m_4-message is not required in Σ_0, since the client Diffie-Hellman share g^c is sent already in message m_1 (before m_2!), and thus fin_S can be contained in m_2.

We stress that one could make TLS-DHE provably secure under DDH by making it more similar to Σ_0:

1. Include the client Diffie-Hellman share g^c in
   - the first message m_1 of TLS-DHE and
   - in the signature sent from client to server.
2. Include all data in m_4 of TLS-DHE into message m_2 of TLS-DHE, and omit m_4.

This modification would allow to carry the security analysis of Σ_0 from [20] over to TLS-DHE, and thus allow a security proof under DDH instead of PRF-ODH (and the additional standard assumptions on the security of the PRF, the signature scheme, and (considering full TLS-DHE) the stateful encryption scheme).

This would of course require changes to the TLS-DHE protocol, and may therefore be unrealistic. This section should therefore merely be seen as an additional discussion of the issue analysed in this section.

INCLUDING IDENTITIES IN THE CLIENT NONCE. Another way (the 'engineering-approach') to circumvent the problem, and thus allow a proof under the DDH assumption instead of PRF-ODH, is to modify TLS such that the client C is able to verify that the server has indeed intended partner C, and not some third party C'. Note that the client nonce r_C is included in both signatures, in particular in the signature σ_S sent from the server to the client. According to [25, 26, 27] the nonce r_C consists of 28 random bytes (=224 bits). The only requirement on the nonces in the proof is that a collision occurs with sufficiently small probability, for which 160 bits should be sufficient in practice. One could use the remaining 64 bits to encode an 'identity' that refers uniquely to the client certificate. The server would have to check whether this identity matches the received client certificate. This would, again, require changes to the TLS-DHE protocol, and may therefore be unrealistic, even though these changes are minimal.

9 Conclusion

In this paper we have shown that the core cryptographic protocol underlying TLS with ephemeral Diffie-Hellman (DHE) provides a secure establishment of confidential and authenticated channels. Contrary to what previous analyses might suggest the random oracle model is not required to show that the composition of cryptographic building blocks in TLS is secure, if we make a suitable assumption on the pseudo-random function.

The design of the TLS-DHE Handshake seems to support our proof idea of reducing active adversaries to passive attackers very naturally. This is not due to the key transport mechanism itself but rather to the fact that DHE uses signatures computed over all previously exchanged messages to authenticate the protocol parties. Our proof essentially exploits that this authentication mechanism at the same time also protects the first phase of the TLS-DHE Handshake from adversarial modifications. We cannot identify a similarly straight-forward approach for encrypted key transport, as server authentication is done rather implicitly when verifying the `Finished` messages. Likewise, the static Diffie-Hellman key exchange does not use signatures over the first Handshake messages and our proof technique does not apply.

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 TLS Handshake, or to resume sessions using an abbreviated TLS Handshake. We need to leave an analysis of these features for future work, since the complexity of the protocol and security model grows dramatically.

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. Another important open problem is to consider cross-protocol attacks, that exploit for instance possible subtle connections between different ciphersuites. While the example of [55] is impractical (and in this case rather an implementational issue), there may be other sophisticated attacks.

So clearly the security analysis of TLS is not finished yet, there are still many open questions. However, we consider our results as a strong indicator for the soundness of the TLS-DHE protocol. 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] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, April 2001.

[2] Adam Langley, 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] Boaz Barak, Yehuda Lindell, and Tal Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004. `http://eprint.iacr.org/`.

[4] Gregory V. Bard. The vulnerability of SSL to chosen plaintext attack. Cryptology ePrint Archive, Report 2004/111, 2004. `http://eprint.iacr.org/`.

[5] Gregory V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. In Manu Malek, Eduardo Fernández-Medina, and Javier Hernando, editors, *SECRYPT*, pages 99–109. INSTICC Press, 2006.

[6] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 1–11. ACM Press, November 2002.

[7] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. *ACM Trans. Inf. Syst. Secur.*, 7:206–241, May 2004.

[8] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, December 2000.

[9] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008.

[10] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, May 2000.

[11] 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.

[12] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, May / June 2006.

[13] Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zalinescu. Cryptographically verified implementations for TLS. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS*

08: 15th Conference on Computer and Communications Security, pages 459–468. ACM Press, October 2008.

[14] 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.

[15] 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.

[16] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer-Verlag, Berlin,Germany, 2003.

[17] C. Brzuska, M. Fischlin, N.P. Smart, B. Warinschi, and S. Williams. Less is more: Relaxed yet composable security notions for key exchange. Cryptology ePrint Archive, Report 2012/242, 2012. `http://eprint.iacr.org/`.

[18] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society Press, October 2001.

[19] 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.

[20] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, August 2002. `http://eprint.iacr.org/2002/120/`.

[21] S. Chaki and A. Datta. Aspier: An automated framework for verifying security protocol implementations. In *Computer Security Foundations Symposium, 2009. CSF '09. 22nd IEEE*, pages 172 –185, july 2009.

[22] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, December 2005.

[23] Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier. New attacks on PKCS#1 v1.5 encryption. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 369–381. Springer, May 2000.

[24] Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09: 7th International Conference on Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 20–33. Springer, June 2009.

[25] 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.

[26] 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.

[27] 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.

[28] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[29] D. Eastlake 3rd and T. Hansen. US Secure Hash Algorithms (SHA and HMAC-SHA). RFC 4634 (Informational), July 2006.

[30] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFC 4634.

[31] 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.

[32] 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 *LNCS*, pages 313–327. Springer, 2008.

[33] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, August 2001.

[34] 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.

[35] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.

[36] Eike Kiltz, Adam O'Neill, and Adam Smith. Instantiability of RSA-OAEP under chosen-plaintext attack. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 295–313. Springer, August 2010.

[37] Eike Kiltz and Krzysztof Pietrzak. On the security of padding-based encryption schemes - or - why we cannot prove OAEP secure in the standard model. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 389–406. Springer, April 2009.

[38] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, August 2001.

[39] 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.

[40] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. In *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 41–50. ACM Press, 2011.

[41] 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*, volume 4784 of *LNCS*, pages 1–16. Springer, 2007.

[42] Gary Locke and Patrick Gallagher. FIPS PUB 186-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS), 2009.

[43] Ueli Maurer and Björn Tackmann. On the soundness of authenticate-then-encrypt: Formalizing the malleability of symmetric encryption. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 505–515. ACM Press, October 2010.

[44] John C. Mitchell. Finite-state analysis of security protocols. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *LNCS*, pages 71–76. Springer, 1998.

[45] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73. Springer, December 2008.

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

[47] Kazuhiro Ogata and Kokichi Futatsugi. Equational Approach to Formal Analysis of TLS. In *ICDCS*, pages 795–804. IEEE Computer Society, 2005.

[48] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *Advances in Cryptology – ASIACRYPT 2011*, Lecture Notes in Computer Science, pages 372–389. Springer, December 2011.

[49] Lawrence C. Paulson. Inductive Analysis of the Internet Protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[50] David Pointcheval and Serge Vaudenay. On Provable Security for Digital Signature Algorithms. Technical report, Ecole Normale Superieure, 1996.

[51] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992.

[52] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, Nov 2004.

[53] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*, pages 19–30, 2002.

[54] Serge Vaudenay. The Security of DSA and ECDSA. In *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *LNCS*, pages 309–323, 2003.

[55] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 29–40. USENIX Association, 1996.

[56] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *IEEE Symposium on Security and Privacy*, pages 35–49. IEEE Computer Society, 2008.

[57] William Zeller and Edward W. Felten. Cross-site request forgeries: Exploitation and prevention. Technical report, October 2008. Available at http://from.bz/public/documents/publications/csrf.pdf.

A Choosing the Right Model

Authenticated key exchange (AKE) is a basic building block in modern cryptography. Many secure protocols for two-party and group key agreement have been proposed, including generic compilers that transform simple key agreement protocols into authenticated key agreement protocols, with many additional security properties. However, since many different formal models for different purposes exist, choice of the right model is not an easy task, and must be considered carefully.

The main guideline for this choice is the fact that we cannot modify any detail of the TLS protocol, nor of the network protocols preceding it.

First, we need a model were *entity authentication* is addressed as a security goal. This goal is often omitted in newer models, in order to make them suitable for two-party authenticated key agreement protocols [39]. However, explicit authentication is an important security goal for TLS, in many practical applications authentication is more important than encryption. For example, in a Single Sign-On scenario, an encrypted security token may be passed from the identity provider through the browser to a relying party. Since the security token itself is encrypted, confidentiality is not an issue, but the authenticity of the channel through which this token was received is crucial.

Second, there is no way to modularize the security proof of TLS in the sense of [19], since several protocol messages of TLS come without authenticator. Thus we cannot use the authenticated link model (AM).

Third, we have chosen not to use a Universal Composability (UC) [18] approach. We think that a formalization in the UC model first requires a thorough analysis in the standard model. Since the exchange of nonces r_C and r_S in the first two messages of the TLS Handshake can be regarded as an instantiation of the Barak compiler [3], it seems in principle possible to model TLS within the UC framework.

On the other hand, we have to make a choice about the enhanced adversarial capabilities newer models offer. We allow for RevealKey queries, but do not take into account RevealState queries. The reason for this omittance is that in TLS there are several successive internal states: Computation of the premaster secret, computation of the master secret, computation of the session keys. After transition from one state to another, internal data is erased. So to be precise, we would have to specify several different RevealState queries, which would have added tremendous complexity to both the model and the proof and rendered the paper unreadable.

Thus we have chosen in essence the first model of Bellare and Rogaway [11], enhanced with adaptive corruptions and perfect forward secrecy. Similar variants of this model have been used by [14, 19], and

especially by [45].