On the Security of TLS-DH and TLS-RSA in the Standard Model[1]

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

Sven Schäge[2]
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 17, 2013

Abstract

TLS is the most important cryptographic protocol in the Internet. At CRYPTO 2012, Jager *et al.* presented the first proof of the unmodified TLS with ephemeral Diffie-Hellman key exchange (TLS-DHE) for mutual authentication. Since TLS cannot be proven secure under the classical definition of authenticated key exchange (AKE), they introduce a new security model called authenticated and confidential channel establishment (ACCE) that captures the security properties expected from TLS in practice. We extend this result in two ways. First we show that the cryptographic cores of the remaining ciphersuites, RSA encrypted key transport (TLS-RSA) and static Diffie-Hellman (TLS-DH), can be proven secure for mutual authentication in an extended ACCE model that also allows the adversary to register new public keys. In our security analysis we show that if TLS-RSA is instantiated with a CCA secure public key cryptosystem and TLS-DH is used in scenarios where a) the knowledge of secret key assumption holds or b) the adversary may not register new public keys at all, both ciphersuites can be proven secure in the standard model under standard security assumptions. Next, we present new and strong definitions of ACCE (and AKE) for server-only authentication which fit well into the general framework of Bellare-Rogaway-style models. We show that all three ciphersuites families do remain secure in this server-only setting. Our work identifies which primitives need to be exchanged in the TLS handshake to obtain strong security results under standard security assumptions (in the standard model) and may so help to guide future revisions of the TLS standard and make improvements to TLS's extensibility pay off.

Keywords: authenticated key exchange, SSL, TLS, provable security, static Diffie-Hellman, RSA, encrypted key transport

[1] The following paper is a full version. An extended abstract of this paper has been rejected at CRYPTO'13.
[2] Supported by EPSRC grant number Grant EP/J009520/1.

Contents

1 Introduction **3**

2 Preliminaries and Definitions **9**
- 2.1 The Decisional Diffie-Hellman Assumption . . . 9
- 2.2 Digital Signature Schemes . . . 9
- 2.3 Public Key Encryption Schemes . . . 9
- 2.4 Pseudo-Random Functions . . . 10
- 2.5 Collision-Resistant Hash Function . . . 11
- 2.6 The (Strong) PRF-Oracle-Diffie-Hellman Assumption . . . 11
- 2.7 Stateful Length-Hiding Authenticated Encryption . . . 12

3 Transport Layer Security (TLS) 1.2 **13**

4 AKE Protocols **17**
- 4.1 Execution Environment . . . 17
- 4.2 Security Definition . . . 19
- 4.3 Server-Only Authentication . . . 20

5 ACCE Protocols **22**
- 5.1 Execution Environment . . . 22
- 5.2 Security Definition . . . 23

6 TLS-RSA with Server-Only Authentication is ACCE-SO Secure **25**
- 6.1 Server-Only Authentication . . . 26
- 6.2 Indistinguishability of Keys . . . 31

7 TLS-DH with Mutual Authentication is ACCE Secure **34**
- 7.1 Client Authentication . . . 35
- 7.2 Server Authentication . . . 39
- 7.3 Indistinguishability of Keys . . . 39

8 Proof Sketches **42**
- 8.1 Proof Sketch for Mutually Authenticated TLS-RSA . . . 42
- 8.2 Proof Sketch for Server-Only Authenticated TLS-DHE . . . 43
- 8.3 Proof Sketch for Server-Only Authenticated TLS-DH . . . 45

References **47**

1 Introduction

TRANSPORT LAYER SECURITY (TLS) is the most important security protocol in the Internet. Despite several efforts, none of the existing security analyses of TLS is truly satisfactory. They have either only examined modified variants of TLS, given proofs in weak security models, or established security results under very strong security assumptions. Only recently Jager, Kohlar, Schäge, and Schwenk (JKSS) proposed the first analysis of the unmodified core of TLS with ephemeral Diffie-Hellman key exchange (TLS-DHE) in a strong security model that also covers replay and re-ordering attacks [30]. To gain provable security results for the unmodified TLS handshake, they did not view TLS as an authenticated key exchange (AKE) protocol, a security notion which TLS provably fails to meet, but as a so-called authenticated and confidential channel establishment (ACCE) protocol. This reflects the application of TLS well, as TLS is not used to provide keys for arbitrary applications (unlike what is expected from traditional key exchange protocols) but only for a dedicated encryption system that encrypts all data exchanged between client and server, starting with the Finished messages of the TLS handshake. Somewhat surprisingly, the JKSS security proof does not necessarily rely on random oracles: if the basic primitives of TLS are substituted by schemes that are secure in the standard model, the entire security proof does not require random oracles. However, due to the design of TLS-DHE, the proof seemingly cannot be based on the standard DDH assumption. This is why Jager *et al.* introduce a new complexity assumption, the PRF-ODH (PRF Oracle Diffie-Hellman) assumption (which is a variant of the Oracle Diffie-Hellman assumption proposed in [1]), that they need to rely on in the security proof. The JKSS proof is for mutual authentication (only) which assumes that all clients also have (and use) certificates when communicating with servers.

To us, the JKSS approach of considering the *unmodified* handshake in a model *without idealized setup assumptions (i.e. random oracles)* seems to be the most realistic and meaningful for the analysis of (practical implementations of) TLS. However, the JKSS result leaves open several important questions. Probably the most important is, which security guarantees are provided by TLS-RSA with server-only authentication, the most popular ciphersuite in the Internet.

Contribution. In this work we extend the JKSS result in two directions. First, we analyze the remaining ciphersuite families that are based on either RSA key transport (TLS-RSA) or static Diffie-Hellman (TLS-DH). We use a slightly enhanced (and more realistic) security model in which each oracle is not given all public keys of the other parties at setup, but where public keys are obtained from the certificates that are exchanged in the TLS protocol run. At the same time we give the adversary extended attack capabilities by allowing him to adaptively request certificates on arbitrary public keys when providing a corresponding proof of knowledge of the secret key. Second, we introduce security notions of AKE and ACCE for server-only authentication and show that all TLS handshake families provide ACCE security in this setting. Our results cover the practically most important ciphersuites of TLS, `TLS_RSA_*` with server-only authentication, including the only mandatory ciphersuite in TLS 1.2, `TLS_RSA_WITH_AES_128_CBC_SHA`.

Limitations. Our results do have some important limitations which we want to point out here. First, for TLS-RSA *we need to require that the underlying public key encryption system is CCA secure*. This stands in line with previous works like [26, 14]. However the TLS-RSA ciphersuites all rely on RSA-PKCS#1 v1.5 which is not CCA secure [13]. Strictly speaking our result does thus not apply to the current state of TLS-RSA. However, we believe that our results are still meaningful and useful for the understanding of TLS and the future development of the TLS standard. They show that changing the encryption system to a provably secure one pays off in the sense that it allows to obtain a strong security result for the *entire* TLS protocol.

We stress that although standardized in PKCS#1 v2.1, TLS does *not* allow to use RSA-OAEP [8] (which in turn is CCA secure under the RSA assumption in the random oracle model only [25, 34] - in the standard model it is only known to be CPA secure under the Φ-Hiding assumption [33] and the assumption that the employed hash functions are t-wise independent) because of 'maximal compatibility with earlier versions of TLS' [22]. We therefore do not follow [26] and rather assume a generic CCA secure encryption scheme. In our security model, it seems necessary that the encryption scheme is CCA secure and thus probabilistic since we need to simulate the correct behaviour of the server that may have to decrypt ciphertexts generated by the adversary. This stands in contrast to the results of Morrissey, Smart, and Warinschi (MSW) [40] who can rely on a much weaker notion of security of the encryption system, OW-CPA security, that for example is fulfilled by (deterministic) text-book RSA. To us, their result seems to be closely tied to the choice of their security model and to the fact that they rely on random oracles in the security proof in a crucial way. By this we mean that their proof would have to rely on random oracles *even* if all primitives used were secure in the standard model. (In this context we note that the informal argumentation of MSW for the necessity of the random oracle model in their TLS-RSA security proof – that a security proof in the standard model would imply a CCA secure encryption scheme under the RSA assumption, what was believed to be a hard, long-standing open problem at the time of publication of [40] – has been invalidated recently by Hofheinz and Kiltz which presented a CCA secure encryption scheme that is secure under the factoring assumption which is weaker than the RSA assumption [28].)

In contrast, our results show that if we replace the encryption system and the signature scheme with primitives provably secure in the standard model[1], we can obtain provably secure results for TLS that avoid random oracles at all. This again can guide future revisions of the TLS standards and distinguishes our result not only from [41] but also from the recent analysis of Brzuska *et al.* [14]: both works use the random oracle to efficiently turn computational security guarantees into decisional ones as sketched below. By modeling the key derivation functions of TLS as random oracles in the security proof, Brzuska *et al.* can easily avoid the simulation problems which made JKSS introduce the non-standard PRF-ODH assumption and which – without neither the PRF-ODH assumption nor random oracles – seem to be hard to solve.

It is obvious that, for compatibility reasons, the primitives in TLS should not be simply exchanged with new ones. In general, we support a careful transition to a more modular structure of TLS in line with one of the most important goals of the TLS protocol given in the specification of TLS 1.2 [22] – extensibility. The standard is specifically explicit with respect to the support of new public key encryption mechanisms. 'Extensibility: TLS seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary'. We emphasize that in the past there have already been modifications of TLS that tend into this direction. For example, while the PRF used in TLS 1.1 is fixed for all ciphersuites, TLS 1.2 allows to use 'cipher-suite-specified PRFs'. Similarly, we might regard the introduction of elliptic curve cryptography (ECC) into the SSL/TLS protocol stack as a positive example of an extension of the TLS standard [11]: more and more servers support and use the efficient ECC-based ciphersuites as default, the most prominent example probably being Google that switched to ECDHE in 2011, to support and push the use of TLS handshakes with forward secrecy [2]. To us, our results not only propose (and analyse) an alternative instantiation of TLS with different primitives. They rather help to lead the process of selecting new, adequate building blocks into a wortwhile direction (towards provable security in the standard model under standard assumptions).

Strategy: From AKE to ACCE. Strategically, our results follow the same outline as JKSS. We (implicitly) first show that the truncated handshake variants of the considered ciphersuite is (server-only) AKE

[1]Natural candidates secure under the RSA assumption are [28] for public-key encryption and [29] for signature generation.

secure. In the next step this result is used to show ACCE security of the full TLS handshake. Let us briefly recall the difference between the AKE and ACCE definitions. In the AKE definition the adversary is given a challenge key and the adversary has to decide whether it is a random key or the real key established in the protocol execution. This where the proof of the *full* TLS protocol breaks down: in the full protocol there always exist some check values, encryptions of known plaintext bytes that the adversary can try to decrypt using its challenge key, which can give the adversary information on whether the key is random or not. In the ACCE model however, these problems do not occur since no challenge key is ever given to the adversary. Instead the key is directly used to key a (stateful) symmetric encryption system. Now the adversary wins the security game if it can distinguish between *encryptions* of two distinct messages or produce new ciphertexts without using the encryption oracle. We stress that the authentication requirements of the AKE and ACCE definitions are equivalent in both the mutual authentication and the server-only authentication setting.

Comparison to the Proof of TLS-DHE. On a technical level there are some crucial differences in the proofs of TLS-RSA and TLS-DH as compared to TLS-DHE. In particular, we cannot rely on the security of a signature scheme to protect the random nonces exchanged in the first protocol phase from adversarial modifications (see Section 3 for a full description of the protocol). Instead, we have to exploit that the Finished messages, which are generated by a pseudo-random function, are computed over all messages exchanged between client and server. Second, in TLS-DHE we can embed the DDH (PRF-ODH) challenge in the Diffie-Hellman shares that are generated freshly each session. In contrast, in TLS-DH the challenge has to be embedded in the long-term key(s) of a party. As a consequence we have to rely on a slightly strengthened PRF-ODH assumption in the security proof of TLS-DH (with server-only authentication) , where a) the number of oracle queries granted to the adversary is only polynomially bounded instead of being constant to 1 and b) the set of input values that can be sent to the PRF-ODH oracle is less restricted. Similarly, in TLS-RSA the challenge of the public key encryption system (PKE) is embedded in the long-term key of the server. Third, we cannot have perfect forward secrecy as in TLS-RSA and TLS-DH the corruption of the long-term key of a server would enable the adversary to decrypt all previous communications with that server. In TLS-DH with mutual authentication (but not in TLS-RSA) this problem also occurs when only client certificates are corrupted. To us this seems to be a serious drawback of the non-DHE ciphersuites.

Security Proofs under Standard Assumptions. Some of our results are surprisingly positive. Similar to the JKSS result, the proof of TLS-DH (with server-only authentication) relies on the (slightly strengthened) PRF-ODH assumption. However, for mutual authentication we can get a proof under the standard DDH (Decisional Diffie-Hellman) assumption in scenarios where the knowledge of secret key assumption (KOSK) holds (which basically means that the adversary always hands over the corresponding secret key when querying a certificate on a public key) or where the adversary cannot register new public keys at all. In practice one could realize the KOSK assumption by forcing the CA to require extractable zero-knowledge proofs of knowledge of the secret key before certifying public keys, for example by using GOS proofs[2] [27] or alternatively [23] in the random oracle model[3]. Anyway, such techniques are not standardized and much more expensive than the current techniques defined in PKCS#10 [42] and RFC 4210/11 [3, 47]. Also, implementing the KOSK in practice does seem to require considerable modifications of current browsers [6]. Applications where the adversary cannot register public keys (which are more realistic in practice) include closed systems where the CA only certifies keys that have been generated in a controlled and trustworthy environment, for example when triggering a key generation algorithm on a smart card that is later given to

[2]A similar approach has been proposed in [38].

[3]Applying NIZKs would additionally require a CRS to be available to all parties.

users of the system (as in Pay TV), or in systems with a fixed user base. Note that in TLS-DHE the security proof is essentially independent of the registration process of long-term keys and consequently these issues do not need to explicitly be taken care of in the original security model of JKSS. [4]

A second positive result of our analyses is that the security proof for TLS-RSA does *not* require to rely on new, non-standard complexity assumptions (unlike the Diffie-Hellman based ciphersuites that rely on the non-standard PRF-ODH assumption). So, when using a CCA secure encryption system and assuming that the underlying signature scheme is existentially unforgeable under adaptive chosen message attacks, the cryptographic core of TLS-RSA is provably ACCE secure.

Server-Only Authentication. Our formalization of server-only authentication fits well into the original Bellare-Rogaway framework of security models [7] and its variants. In particular, we stick to the theoretically simple and clean concept of matching conversations to define authentication between communication partners without requiring the existence of session IDs prior to the protocol execution. We provide a comparatively strong definition of server-only authentication. It guarantees that – despite the absence of an authentication mechanism for the client – the client only accepts if *none* of the exchanged messages *including those sent from the client to the server* has been modified, meaning that client and server have *matching conversations* in the traditional sense. In particular, we do not confine ourselves with protocols where the client is only guaranteed that the messages sent from the server to the client have not been modified in transit. For clarity, let us point out the difference when analysing protocols with server-only authentication as compared to protocols with mutual authentication more explicitly as both define authentication using matching conversations. In the latter (one can technically exploit that) not only the server but also the client can use an authentication mechanism that is keyed with a long-term key. This is not possible when proving server-only authentication. It is, for example easy to see that a protocol where client and server additionally sign each of their messages before sending them provides mutual authentication: any attacker that makes client or server accept without a matching conversation can be used to break the security of the signature scheme. However, the same protocol does in general not meet our strong notion of server-only authentication when the client looses its authentication mechanism: an adversary may simply alter one of the messages sent from the client to the server. Since theses messages are not checked for authenticity the sender or receiver might accept without having a matching conversation while there is no way to reduce such an attacker to any underlying security assumption. Intuitively, in protocols that meet our server-only definition there must be a mechanism for the client to check whether the server received all the messages sent by the client without any modification.

Looking somewhat ahead, TLS does fulfill our strong definition of server-only security. Practically, this fits well to most of the applications of TLS, as the client is usually interested in also knowing that the session keys computed by the server are secure and all the (confidential) data sent from the server to the client is protected as well. From another perspective, the server can entirely make it the client's responsibility to check if no message has been modified in transit. The technical mechanism which signals the client that all its messages have been received by the server without modification is the (encrypted) server finished message that contains a MAC (which is implemented via a PRF) over the entire transcript so far.

Although the authentication guarantees of a server-only AKE protocol may be different, we need the same strong key indistinguishability guarantees as in AKE protocols with mutual authentication. The main

[4] In TLS-DHE all long-term keys are only used to compute signatures over transcripts. When embedding the signature challenge one can easily simulate the behaviour of an uncorrupted party by using the signing oracle of the EUF-CMA security game to sign adversarially chosen messages.) In contrast, in our extended ACCE (and AKE) model (which should allow to properly model TLS-RSA and TLS-DH too), we allow the adversary to certify new public keys and account for attackers that break the signature scheme used in the certification process (by the certification authority.

difference is that now the Test query can only be issued to clients.

Stateful Length Hiding Encryption. We rely on the recent results of Paterson, Ristenpart and Shrimpton who showed that the CBC-based Record Layer protocols of TLS 1.1 and 1.2 provably guarantee strong security properties of the data encryption mechanism of TLS [44]. Concretely they showed that the TLS Record Layer meets the definition of length hiding authenticated encryption (LHAE). We use the stateful variant of this definition (stateful LHAE or sLHAE) that also protects against drops, replay attacks, or re-ordering attacks.

Related Work. The literature contains a plethora of works on the security of TLS. These works range from the publication of minor flaws and new attacks against SSL 3.0 [49] to detailed analyses of the asymmetric encryption system used in TLS-RSA and PKCS#1 [13, 19, 32, 34, 33], automated proof techniques [39, 45, 43, 10, 18] (which are not known to be cryptographically sound) and the analyses of the single primitives used in TLS [35, 44]. In all of these works there is a general tendency towards modeling TLS in more and more detail.

Arguably the first approach to analyze the security of the entire TLS handshake in a cryptographic model is that of Johnsson and Kaliski [32]. They provide a positive result on the use of PKCS#1 v1.5 encryption under a strong new complexity assumption that is related to RSA which requires that a partial decryption oracle exists. Under this assumption, they showed that the key transport mechanism of TLS-RSA is secure if the interaction between the client and the server is modeled as a tagged key-encapsulation mechanism (TKEM). They only consider a simplified version of the handshake that omits the Finished message of the server.[5] The TKEM security model is a considerable simplification of the 'usual' execution environment in models for analyzing cryptographic protocols where not only parties but also distinct executions of a protocol – oracles – are modeled. The TKEM model does also not (explicitly) provide important attack capabilities like the corruptions of secret keys. We remark that in a model without corruptions the simulation problems of JKSS which lead to the introduction of the PRF-ODH assumption simply do not occur.

Gajek *et al.* [26] presented an analysis of the *truncated* versions of all three handshakes in a security model that is formulated in the UC framework of Canetti [16]. They do not use random oracles. However, they consider a modified TLS handshake that enables them to avoid the TLS-DHE simulation problems of JKSS. The Gajek *et al.* result does not consider message replay, drop, or re-ordering attacks. The ideal functionality of the key exchange part is very weak and rather unrealistic: it only formalizes non-adaptive corruptions and only *unauthenticated* key exchange. Their communication channel functionality thus does only cover confidentiality of data but no entity authentication. We believe that this is not appropriate to model TLS which is used as a tool for authenticating communication partners in practice.

Morrissey, Smart and Warinschi (MSW) presented an analysis of the truncated handshakes of TLS in the random oracle model [40, 41]. They define three, progressively stronger security notions and show that the TLS protocols are secure in their model. As pointed out above their result strongly relies on the properties of random oracles: they use random oracles to a) efficiently switch from computational to decisional security guarantees (similar to Kudla and Paterson [37]) and b) to derive the master secret and the application keys and so avoid the simulation problems when dealing with TLS-DHE. The first strategy, a), allows to avoid the problems that occur when trying to prove the *full* handshake AKE secure. MSW do not require that the master key is indistinguishable from random but only that it is not computable by the adversary. This is possible even when the adversary is given access to the encrypted Finished messages. Via the random oracle

[5] They justify this step by claiming that the Finished message of the server 'in any case is no more helpful to an adversary than the one computed by the client'.

they can then easily turn the master secret into encryption keys that are indistinguishable from random. This allows them to only rely on the Gap-CDH assumption when proving TLS-DHE. Similarly to Gajek *et al.*, their result does not consider message replay, drop, or re-ordering attacks. Also, the MSW result does assume that the order of the messages of the TLS handshake is slightly modified.

As sketched before, JKSS [30] introduced ACCE, a new security notion for establishing authenticated and confidential communication channels and proved that TLS-DHE with mutual authentication is ACCE secure with perfect forward secrecy. They show that TLS-DHE with mutual authentication is a secure ACCE protocol under the non-standard PRF-ODH assumption and standard assumptions on the remaining primitives. In particular, their security proof does not essentially rely on random oracles: they point out that when the primitives are instantiated in the standard model (this mainly concerns the signature scheme), the security of the entire TLS-DHE handshake holds in the standard model as well. To us, this gives valuable information in what directions the revision of the TLS standard should be guided. Also, it might for example be possible to introduce a new signature scheme secure without random oracles to be used in TLS via an appropriate standard extension *without modifying the entire TLS stack*. This can lead to a provably secure result in the standard model with minimal modifications. However, as pointed out above the JKSS result only covers the less widespread ciphersuites `TLS_DHE_*` and only mutual authentication.

Recently Brzuska *et al.* [15] presented new security definitions for key exchange protocols which are weaker than AKE but still generally composable with symmetric primitives. They claim that the JKSS security model can be cast as an instantiation of their abstract framework. As in MSW, their proof does essentially rely on random oracles to model how TLS derives the master key and the application keys. Technically, this allows for example to reduce the security of TLS-DHE to the *Computational* Diffie-Hellman (CDH) assumption, as opposed to JKSS who (at least partly) required the stronger Decisional Diffie-Hellman and PRF-ODH assumptions.[6] In contrast to our work, they do not cover server-only authentication (and it is not clear if their composability guarantees also hold in the server-only setting).

To sum up the assumptions underlying previous works, both Brzsuka *et al.* and MSW crucially rely on the random oracle model which in turn has been criticized fundamentally in several works starting with [17]. JKSS showed that the non-standard PRF-ODH assumption is sufficient to prove TLS-DHE with mutual authentication. We show that in special cases even the DDH assumption is sufficient for TLS-DH with mutual authentication and no further non-standard assumption is required when proving TLS-RSA with mutual or server-only authentication when relying on a CCA secure encryption system.

[6]When using random oracles to derive keys, the output of the random oracle is indistinguishable from random if only a single input, for example the premaster secret Diffie-Hellman value g^{cs}, where g^c is the client and g^s the server share, is hard to compute for the adversary. When using a PRF the output is only distributed indistinguishable from random if the key g^{cs} is indistinguishable from random in the first place. This allows to use the CDH assumption in contrast to the DDH assumption in the security proof when relying on random oracles.

2 Preliminaries and Definitions

In this section, we present the formal definitions required to formulate our results. We use $\emptyset$ to denote the empty string, and $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for the set of integers between 1 and n. If A is a set, then we use $a \xleftarrow{\$} A$ to denote that a is drawn uniformly random from A. In case A is a probabilistic algorithm $a \xleftarrow{\$} A$ is used to denote that A returns a when executed with fresh random coins. We use κ to denote the security parameter.

2.1 The Decisional Diffie-Hellman Assumption

Let G be a group of prime order q (having bit-length polynomial in κ) and g a generator of G. The decisional Diffie-Hellman (DDH) assumption states that if given (g, g^a, g^b, g^c) for $a, b, c \in \mathbb{Z}_q$ it is hard to decide whether $c = ab \bmod q$. More formally:

Definition 1. We say that the DDH problem is (t, ϵ)-*hard* in G, if for all adversaries $\mathcal{A}$ that run in time t it holds that

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

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 the 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 the security parameter κ, the signing algorithm $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$ generating a signature for message m, and the verification algorithm $\mathsf{SIG.Vfy}(pk, \sigma, m)$ returning 1, if σ is a valid signature for m under key pk, and 0 otherwise. Security if formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates an asymmetric key pair $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$ and the public key pk is given to the adversary.
2. The adversary may adaptively query q messages m_i with $i \in [q]$ of his choice to the challenger. The challenger responds to each of these queries with a signature $\sigma_i = \mathsf{SIG.Sign}(sk, m_i)$ on m_i.
3. The adversary outputs a message/signature pair (m, σ).

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

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

2.3 Public Key Encryption Schemes

A public key encryption (PKE) scheme is a triple $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, consisting of the key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$ generating a (public) encryption key pk and a secret decryption key sk on input of the security parameter κ, the probabilistic encryption algorithm $\mathsf{PKE.Enc}(pk, m)$

generating a ciphertext c for message m, and the deterministic decryption algorithm $\mathsf{PKE.Dec}(sk, c)$ returning m, if c is a valid encryption and the error symbol $\perp$ otherwise. Security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates an asymmetric key pair $(sk, pk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$ and the public key pk is given to the adversary.
2. The adversary may adaptively query the challenger for decryptions of arbitrary ciphertexts c. The challenger responds to each of these queries with the output of $\mathsf{PKE.Dec}(sk, c)$.
3. The adversary outputs a message m^*.
4. The challenger tosses coin $b \xleftarrow{\$} \{0, 1\}$. It then sets $c_0 = \mathsf{PKE.Enc}(pk, m^*)$ and $c_1 = \mathsf{PKE.Enc}(pk, r)$ for a uniformly random message r that is of the same size as m^* and sends c_b to the adversary.
5. The adversary may again adaptively query ciphertexts c of his choice, now with the restriction that $c \neq c_b$. The challenger responds to each of these queries with the output of $\mathsf{PKE.Dec}(sk, c)$.
6. Finally the adversary outputs a bit b'.

Definition 3. We say that PKE is (t, ϵ, q)-*secure* under *adaptive chosen ciphertext attacks* (CCA), if all adversaries $\mathcal{A}$ that run in time t making at most q decryption queries have advantage of at most ϵ to distinguish the ciphertext of m^* from that of a truly random value, i.e.

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

2.4 Pseudo-Random Functions

A *pseudo-random function* (PRF) is a deterministic algorithm PRF which given a key $k \in \mathcal{K}_{\mathsf{PRF}}$ (with $\log(|\mathcal{K}_{\mathsf{PRF}}|)$ polynomial in κ) and a bit string x outputs a string $z = \mathsf{PRF}(k, x)$ with $z \in \{0, 1\}^\mu$ (and μ being polynomial in κ). Security is formulated via the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$ uniformly random and $b \xleftarrow{\$} \{0, 1\}$.
2. The adversary may adaptively query q values x_i with $i \in [q]$ to the challenger. The challenger replies to each of these queries with either $z_i = \mathsf{PRF}(k, x_i)$ if $b = 0$ or $z_1 \xleftarrow{\$} \{0, 1\}^\mu$ if $b = 1$.
3. Finally, the adversary outputs its guess $b' \in \{0, 1\}$ of b.

Definition 4. We say that PRF is a (t, ϵ, q)-*secure* pseudo-random function, if any adversary running in time t that makes at most q queries has an advantage of at most ϵ to distinguish the PRF from a truly random function, i.e.

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

Remark 1. In our security analyses we rely on the result by Fouque *et al.* [24] who showed that the PRF used in TLS v1.2 is secure with respect to the above definition if the compression function of the hash function used in the HMAC-based key-derivation function of TLS behaves like a pseudo-random function.

2.5 Collision-Resistant Hash Function

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

Definition 5. We say that H is a (t, ϵ)-*secure* collision-resistant hash function, if any t-time adversary $\mathcal{A}$ that is given $k \xleftarrow{\$} \mathcal{K}_{\mathsf{H}}$ has an advantage of at most ϵ to compute two colliding inputs m, m' with $m \neq m'$ and $\mathsf{H}(m) = \mathsf{H}(m')$.

2.6 The (Strong) PRF-Oracle-Diffie-Hellman Assumption

Let G be a group with generator g of order q' (having bit-length polynomial in κ). Let PRF be a deterministic function $z = \mathsf{PRF}(X, m)$ with inputs key $X \in G$ and bit string m that returns a string $z \in \{0,1\}^{\mu}$. Consider the following security game that is played between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. The adversary $\mathcal{A}$ outputs a value m.
2. The challenger samples $u, v \xleftarrow{\$} [q']$ and $z_1 \xleftarrow{\$} \{0,1\}^{\mu}$ 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 adaptively query q pairs (X_i, m'_i) with $X_i \neq g^u$, $i \in [q]$ to the challenger. The challenger replies with $\mathsf{PRF}(X_i^v, m'_i)$.
4. Finally the adversary outputs a guess $b' \in \{0,1\}$.

Definition 6. We say that the (original) PRF-ODH problem is (t, ϵ)-*hard* for G and PRF, if for all adversaries $\mathcal{A}$ that run in time t making at most $q = 1$ query in the above security game it holds that

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

Similarly we say that the Strong PRF-ODH problem is (t, ϵ, q)-*hard* for G and PRF, if for all adversaries $\mathcal{A}$ that run in time t making at most q queries it holds that

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

where in the third step of the above security game we substitute the condition $X_i \neq g^u$ with $(X_i, m'_i) \neq (g^u, m)$.

The definition of the Strong PRF-ODH problem differs from the original definition of JKSS in two ways. First, we allow the adversary to ask a polynomial number of queries to the oracle in contrast to only a single query. This makes our assumption more similar to the Oracle Diffie-Hellman assumption introduced by Abdalla, Bellare and Rogaway in [1] which uses hash functions instead of PRFs. Second, we also allow queries of different messages m'_i with the same key g^u, as long as $m'_i \neq m$. This makes our assumption much more comparable with the classical security definition of PRFs, as now the oracle can also be queried for the challenge key more than once.

$\underline{\mathsf{Encrypt}(m_0, m_1, \mathsf{len}, H)}$:	$\underline{\mathsf{Decrypt}(C, H)}$:
$u := u + 1$	$v := v + 1$
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_0, st_e)$	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. The values u, v and phase are all initialized to 0 at the beginning of the security game.

2.7 Stateful Length-Hiding Authenticated Encryption

We now recall the stateful variant of the definition for length-hiding authenticated encryption (LHAE) from JKSS which is attributed to Paterson *et al.* [44].

A (symmetric) *stateful length-hiding authenticated encryption (sLHAE) scheme* consists of two algorithms $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$. The (possibly probabilistic) encryption algorithm, given as $(C, st'_e) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m, st_e)$, takes the following values as input: the secret key $k \in \{0,1\}^\kappa$, the length $\mathsf{len} \in \mathbb{N}$ of the output ciphertext, header data $H \in \{0,1\}^*$, the plaintext $m \in \{0,1\}^*$, and the current state $st_e \in \{0,1\}^*$ of the encryption scheme. It outputs either a ciphertext $C \in \{0,1\}^{\mathsf{len}}$ and the updated state st'_e or the special error symbol $\perp$. The deterministic decryption algorithm $(m', st'_d) = \mathsf{StE.Dec}(k, H, C, st_d)$ processes secret key k, header data H, ciphertext C, and the current state $st_d \in \{0,1\}^*$. It returns the 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. The encryption state st_e and decryption state st_d are initialized to the empty string $\emptyset$.

Consider the following security game that is played between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. The challenger draws $b \xleftarrow{\$} \{0,1\}$ and $k \xleftarrow{\$} \{0,1\}^\kappa$ and sets $st_e := \emptyset$ and $st_d := \emptyset$,.
2. The adversary may adaptively query the encryption oracle Encrypt q_e times and the decryption oracle Decrypt q_d times. Figure 1 shows how these oracles respond to $\mathcal{A}$'s queries.
3. Finally, the adversary outputs a guess $b' \in \{0,1\}$.

Definition 7. We say that an sLHAE scheme $\mathsf{StE} = (\mathsf{StE.Init}, \mathsf{StE.Enc}, \mathsf{StE.Dec})$ is (t, ϵ)-*secure*, if for all adversaries $\mathcal{A}$ that run in time t it holds that

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

in the above security game. We assume that the number of encryption queries q_e and decryption queries q_d is bound by the running time of the adversary.

3 Transport Layer Security (TLS) 1.2

In this section, we present the core of the current version of the TLS protocol – TLS 1.2 [22]. The major changes from TLS 1.0 [20] and TLS 1.1 [21] to TLS 1.2 include a different error handling procedure to prevent certain attacks (e.g. TLS 1.1 introduced an explicit Initialization Vector and handled padding errors differently in order to prevent CBC attacks) and, perhaps most importantly, switching from a (fixed) MD5/SHA-1 hash function combination in the PRF to ciphersuite-specific hash functions, with SHA-256 being the new standard. The basic protocol flow in TLS is fixed (and the same) for all ciphersuites, differing only in (i) the content of the messages sent and (ii) the set of messages sent depending on the type of authentication (server-only or mutual).

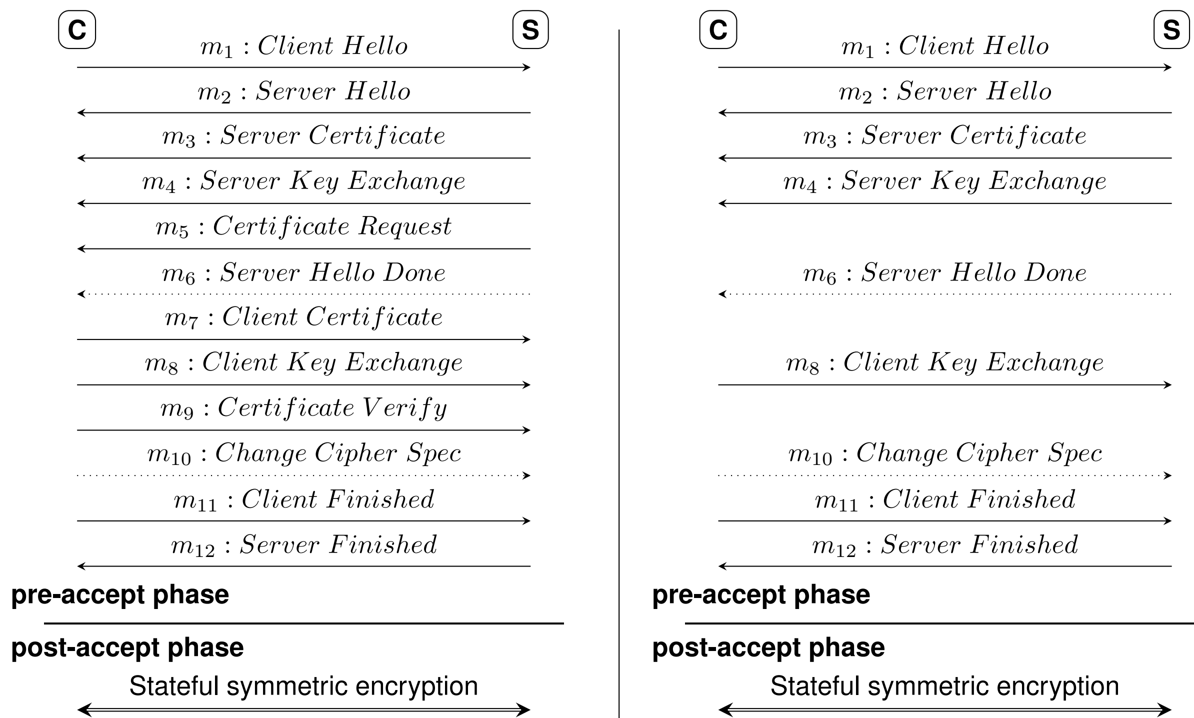

Figure 2: TLS handshake for ciphersuites TLS-DH, TLS-DHE and TLS-RSA with mutual (left) and server-only (right) authentication. Dotted lines mark control messages that contain no cryptographic information.

Figure 2 depicts the message flow of TLS if the server additionally requests client authentication (left side) and the message flow if only server-authentication is required (right side). Depending on whether client authentication is requested, the TLS Handshake Protocol consists of 11, respective 8 messages. The messages can contain cryptographic information as Diffie-Hellman group parameters and public keys as well as constant byte values. In the following, we list all messages sent during a TLS protocol run with mutual authentication and explain their explicit content and function.

m_1 m_2 **Client Hello and Server Hello.** The Hello messages contain a random 32 byte value (nonce) that is used in a later stage to derive the master secret. The nonce sent by the client is denoted as r_C, the one sent by the server as r_S. Note that for the client only 28 bytes are chosen completely at random while 4 bytes are derived from the local time of the client. The Client Hello message also includes a list of ciphersuites supported by the client. The Server Hello message contains the server's choice

of the ciphersuite that is then used by both parties. The Hello messages may also contain additional information such as the compression method applied to the messages before sending.

m_3 **Server Certificate.** This message contains the server certificate $cert_S$ (or a certificate chain), which binds a public key to the owner of the certificate (i.e. the server). Besides the server identity, the certificate may contain a subset of the following information, depending on the ciphersuite used:

- public parameters for a Diffie-Hellman based key exchange (a prime number p, a generator g for a prime-order q subgroup of $\mathbb{Z}_p^*$ and a public Diffie-Hellman key (or "share") g^s (where $s \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ is the corresponding secret key) in TLS-DH,
- a public key pk_S used for encrypted key transport in TLS-RSA or
- a public key pk_S of a signature scheme in TLS-DHE.

Note that the client learns the identity of its communication partner not before receiving this message.

m_4 **Server Key Exchange.** This message is only sent for TLS-DHE and contains public parameters for a Diffie-Hellman based key exchange (as defined in the previous message) along with a signature covering both nonces r_C, r_S and the public parameters.

m_5 **Certificate Request.** By sending this message the server requests the clients to provide a certificate. The message contains a list of valid certificate types that the client may use. *This message is only sent when client authentication is required.*

m_6 **Server Hello Done.** Sending this message, the server tells the client to proceed with the next phase of the protocol and check the validity of the server certificate received in m_3.

m_7 **Client Certificate.** This message contains the client certificate $cert_C$ (or a certificate chain). The certificate contains the client identity and a public key pk_C for a signature scheme. For TLS-DH ciphersuites, the certificate also contains a fixed Diffie-Hellman public key g^c used for computation of the premaster secret pms (where the g is the genenerator specified in the server certificate and $c \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ is the corresponding secret key). Note that the server learns the identity of its communication partner not before receiving this message. *This message is only sent when client authentication is required.*

m_8 **Client Key Exchange.** The client always sends a Client Key Exchange message, the content depends on the negotiated ciphersuite:

- a public DH key g^c (which as before must match the parameters received by the server) in TLS-DHE,
- a 48-byte premaster secret encrypted under the public key of the server in TLS-RSA, or
- an empty string in TLS-DH.

m_9 **Certificate Verify.** Here, the client sends a signature σ_C computed on the concatenation of all messages exchanged so far (that is m_1 to m_8). *This message is only sent when client authentication is required and the client certificate does not contain static Diffie-Hellman keys.*

m_{10} **Change Cipher Spec.** This message consists of a single byte of value '1' and indicates, that subsequent messages will be encrypted under the newly established keys.

m_{11} m_{12} **Client/Server Finished.** The Finished messages contain a stateful encryption of the following information:

$$fin_C := \mathsf{PRF}(ms, label, \mathsf{H}(m_1, \ldots, m_{10})), \text{ and } fin_S := \mathsf{PRF}(ms, label, \mathsf{H}(m_1, \ldots, m_{10}, fin_C{}^7))$$

where $label$ ='client finished' for the Client Finished message fin_C and $label$ ='server finished' for the Server Finished message fin_S, and H denotes a collision-resistant hash function. That is $m_{11} = \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, fin_C, st_e)$ and $m_{12} = \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, fin_S, st_e)$, where $k_{\mathsf{enc}}^{\mathsf{Client}}$, $k_{\mathsf{enc}}^{\mathsf{Server}}$ and ms are defined as below.

	mutual TLS-RSA	server-only TLS-RSA	mutual TLS-DH	server-only TLS-DH	server-only TLS-DHE
Client Hello	r_C	r_C	r_C	r_C	r_C
Server Hello	r_S	r_S	r_S	r_S	r_S
Server Certificate	$cert_S$	$cert_S$	$cert_S, g^s$	$cert_S, g^s$	$cert_S$
Server Key Exchange	-[8]	-	-	-	g^s
Client Certificate	$cert_C$	-	$cert_C, g^c$	-	-
Client Key Exchange	$\mathsf{PKE.Enc}(pms)$	$\mathsf{PKE.Enc}(pms)$	-	g^c	g^c

Figure 3: Ciphersuite-dependent TLS messages

In the following we will explain, how intermediate keys and application keys are derived from the messages exchanged between the two parties.

COMPUTING THE PREMASTER SECRET. The premaster secret pms is either chosen by the client (in TLS-RSA) or computed as g^{cs} for DH-based ciphersuites, where g is a generator for some prime-order subgroup, g^s is the public DH key of the server and g^c the public DH key of the client.

COMPUTING THE MASTER SECRET. The master secret ms is computed by applying the PRF of TLS, keyed with the premaster secret pms, to a fixed label $label_1$ and the nonces exchanged between the parties r_C, r_S.

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

The master secret will then be used to derive application keys and to compute the Finished messages.

COMPUTING THE APPLICATION KEYS. The four application keys (encryption and MAC keys for each direction) are also computed using the PRF of TLS, where the inputs are now the master secret ms, another fixed label $label_2$ and (again) the random nonces r_C, r_S.

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

We also define

$$k_{\mathsf{enc}}^{\mathsf{Client}} = k_{\mathsf{dec}}^{\mathsf{Server}} := K_{\mathsf{enc}}^{C \to S} || K_{\mathsf{mac}}^{C \to S}$$

and

$$k_{\mathsf{enc}}^{\mathsf{Server}} = k_{\mathsf{dec}}^{\mathsf{Client}} := K_{\mathsf{enc}}^{S \to C} || K_{\mathsf{mac}}^{S \to C}.$$

[7]Note, that m_{11} contains an **encryption** of fin_C, and that fin_S is computed over plaintext handshake messages only.
[8]'-' means that the message is not sent (or empty) in this setting

We refer to these application keys as **encryption keys**. In the following we use k_S short for the encryption key(s) $k_{\mathsf{enc}}^{\mathsf{Server}} = k_{\mathsf{dec}}^{\mathsf{Client}}$ and k_C short for the encryption key(s) $k_{\mathsf{enc}}^{\mathsf{Client}} = k_{\mathsf{dec}}^{\mathsf{Server}}$. We think of these encryption keys to key a stateful length-hiding authenticated encryption scheme as defined in Definition 7.

ATTACK FOCUS. We do not consider abbreviated TLS Handshakes[9], nor side-channel attacks (such as the Bleichenbacher attack against PKCS #1.5 [13] or the BEAST (Browser Exploit Against SSL/TLS) attack[10] discussed in [4, 5]). Moreover, we do not consider attacks based on side-channels such as error messages, or implementation issues (e.g. the cross-protocol attack by Schneier and Wagner [49]). For simplicity we also assume that TLS compression is not used, excluding attacks like CRIME [46].

[9]We note that the server can always enforce a full TLS Handshake
[10]We also note, that this vulnerability was fixed in TLS 1.1

4 AKE Protocols

In this section we present an extended AKE definition of [7, 30] that also allows the adversary to register key pairs for new parties via a new query called Register. We also give the first definition for server-only authentication and server-only authenticated key exchange protocols. In the subsequent section we will then transfer these results to the ACCE setting.

As it is usual, we set up a common execution environment which models that the adversary controls the communication network. Technically, we model attack capabilities through queries that the adversary may ask to the execution environment. These queries differ only slightly for AKE and ACCE security. The corresponding definitions for server-only authentication use the same execution environment as for mutual authentication. Only the definition of security deviates from that of mutual authentication.

As mentioned above our model is essentially that of Jager *et al.*, which is derived from the model of Bellare and Rogaway [7] transferred to the public key setting [12] with an additional modeling of adaptive corruptions. In contrast to MSW, Brzuska *et al.* and JKSS we also model public-key certification (in a simplified form) and allow to optionally model perfect forward secrecy (via adaptive corruptions). To us this seems necessary, as depending on the type of registration model, security proofs might need to rely on different assumptions (as it is for example the case when using TLS-DH ciphersuites with mutual authentication).

4.1 Execution Environment

Let $\ell, d, w \in \mathbb{N}$. We consider a set of ℓ parties $\{P_1, \ldots, P_\ell\}$. Each party $P_i \in \{P_1, \ldots, P_\ell\}$ is a (potential) protocol participant having a unique long-term key pair (pk_i, sk_i).[11] Additionally each party P_i has a certificate $cert_i$.[12] The certificate is computed by the execution environment (acting as a certification authority) as a signature over the party's public key pk_i, a *unique* identifier (for example a random string or a serial number) id_i, and some additional information aux_i (that can for example be used to identify the party P_i, like a domain name or an email address). The certificates are computed with the secret key sk of the certification key pair pk, sk. We assume that each party P_i is associated with d oracles $\pi_i^1, \ldots, \pi_i^d$, where each π_i^s represents a process that executes one single instance of the protocol. The oracles $\pi_i^1, \ldots, \pi_i^d$ of party P_i all have access to the same long-term key pair (pk_i, sk_i) and the public key pk of the execution environment. 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.
- $\Pi \in [\ell + w]$[13] holding the intended communication partner, i.e., an index j that points to a globally unique certificate identity.
- Variable $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$.

[11] We stress that this also models scenarios where the set of parties may initially be empty and step-wisely be filled by the environment since the public keys are generated independently of the adversary. It makes no difference if the public keys and certificates are generated at setup or on demand. We only require that the total number of parties is at most ℓ. In practice this for example models situation where the certification authority may have been setup and just begins to issue new certificates to parties.

[12] This is just for simplicity. We could easily extend our model to cover situations where each party may have more than one certificate.

[13] w is the maximal number of additional certificates the adversary may request.

- 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

$$k \neq \emptyset \iff \Lambda = \text{`}\texttt{accept}\text{'}. \quad (1)$$

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 $\rho :=$ Client and responds with the first protocol message. Otherwise it sets $\rho :=$ Server and responds as specified in the protocol.[14]

  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 $\Lambda =$ '`accept`', see (1).

- $\mathsf{Corrupt}(P_i)$: Oracle π_i^1 responds with the long-term secret key sk_i of party P_i.[15] 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 $\Lambda \neq$ `accept`, then it returns some failure symbol $\perp$. Otherwise it flips a fair coin b, samples an independent key $k_0 \stackrel{\$}{\leftarrow} \mathcal{K}$, sets $k_1 = k$ to the 'real' key computed by π_i^s, and returns k_b.

- $\mathsf{Register}(pk', aux, proof)$: This query first checks whether the adversary 'knows' the secret key sk' corresponding to the public key pk' by evaluating the non-interactive proof $proof$ of knowledge of sk'[16]. On failure it outputs an error symbol $\perp$, on success it outputs a certificate, binding the public key pk' and aux[17] to a new globally unique identity id' generated by the execution environment. In general this proof only needs to be sound. However when relying on the knowledge of secret key (KOSK) assumption we require that the proof consists of sk'. One can think of $id' = id_z$ as being

[14] 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.

[15] 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.

[16] We only require that the proof is non-interactive to simplify the model (if a common reference string is required we may assume that it is held by the execution environment and made publicly available). In practice, the concrete implementation of these proofs of knowledge is up to the CA [3] and may also be interactive. We only require that it is secure under concurrent executions. Examples can be found in RFC 4210 [3] and PKCS#10.

[17] We opt to not make the environment 'generate' *all* information (including those used in the certificates requested by the adversary). Our model is slightly stronger as the adversary can freely specify aux as part of the certificate.

associated with an index $z \in \mathbb{N}$. At the beginning, z is initialized to $\ell + 1$. For each call of Register, z is incremented by one. This query can be called at most w times.

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 AKE security. The Register-query enables the adversary to register new public keys that can be used to authenticate adversarial messages.

4.2 Security Definition

To define security we use the classical definition of matching conversations by Bellare and Rogaway [7]. Here we use a slightly more general definition introduced in JKSS. 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 8 (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
- $T_{i,s} = T_{j,t}$ and π_j^t has sent the last message(s).

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) and certificates $cert_i$ for all $i \in [\ell]$. The adversary receives the certificates $cert_1, \ldots, cert_\ell$ as input.

Now the adversary may start issuing Send, Reveal, Corrupt and Register queries, as well as one Test-query. Finally, the adversary outputs a bit b' and terminates.

Definition 9. We say that an adversary (t, ϵ)-*breaks* an AKE protocol with mutual authentication, 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 $\Pi = j$ such that $id_j \in \{id_1, \ldots, id_\ell\}$[18], and
   - P_j is τ_j-corrupted with $\tau_0 < \tau_j$,[19] and
   - there is no unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

[18]This means that the certificate with $id = id_j$ has not been generated by a Register query. Otherwise the adversary may know the corresponding secret key and trivially make π_i^s accept.

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

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 $\Pi = j$ such that $id_j \in \{id_1, \ldots, id_\ell\}$, 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 $\mathcal{A}$ *answers the* Test*-challenge correctly*.

We say that an AKE protocol with mutual authentication is (t, ϵ)*-secure* with perfect forward secrecy, if there exists no adversary that (t, ϵ)*-breaks* it. We say that an AKE protocol with mutual authentication is (t, ϵ)*-secure* (without perfect forward secrecy), if there exists no adversary that (t, ϵ)*-breaks* it while (in the above) not being allowed to ask the $\mathsf{Corrupt}$ query to P_i and P_j after the Test query (i.e. $\mathcal{P}_i$ and P_j are both τ-corrupted with $\tau = \infty$).

Remark 2. Note that the above definition also models key-compromise impersonation (KCI) attacks [12, 36], since we do not require that P_i is uncorrupted. Even if P_i is corrupted, it must be impossible for an adversary to impersonate party P_j to P_i. This is also required in all subsequent security definitions.

4.3 Server-Only Authentication

Our definition of server-only authentication is very similar to that of mutual authentication. The main difference is that we now only concentrate on the client. However, as in the definition of mutual authentication we need to be careful to not include trivial attacks where the adversary simply drops the last (unauthenticated) protocol message[20], as such attacks cannot be reduced to any hard problem. In the following we concentrate on authenticated servers. We stress that this is not a restriction. Our definitions can easily be transferred to model client-only authentication or, more generally, initiator-only and responder-only authentication.

Definition 10. We say that an adversary (t, ϵ)*-breaks* an AKE protocol with server-only authentication, 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 that accepts *maliciously* in the sense of Property 1 of Definition 9, and π_i^s has internal state $\rho = \mathsf{Client}$.

2. $\mathcal{A}$ answers the Test-challenge correctly in the sense of Property 2 of Definition 9 and either

   - oracle π_i^s (to which $\mathcal{A}$ has issued the Test query) has internal state $\rho = \mathsf{Client}$, or
   - oracle π_i^s has internal state $\rho = \mathsf{Server}$, and there exists an oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

[20]For more details we refer to [30].

We say that an AKE protocol with server-only authentication is (t, ϵ)-*secure* with perfect forward secrecy, if there exists no adversary that (t, ϵ)-*breaks* it. We say that an AKE protocol with server-only authentication is (t, ϵ)-*secure* (without perfect forward secrecy), if there exists no adversary that (t, ϵ)-*breaks* it while (in the above) not being allowed to ask the $\mathsf{Corrupt}$ query to P_i and P_j after the Test query (i.e. $\mathcal{P}_i$ and P_j are both τ-corrupted with $\tau = \infty$).

Remark 3. Client-only security can be defined analogously, only the internal states ρ of π_i^s and π_j^t are switched. It is then easy to see that if π_i^s has a matching conversation with π_j^t when only allowing server authentication and π_j^t has a matching conversation with π_i^s when only allowing client authentication then π_i^s and π_j^t have matching conversations and the protocol is ACCE secure. However, the converse is not true in general.

Remark 4. Note that our security definitions for protocols without perfect forward secrecy are slightly stronger than the original notions in [31], in that we now also restrict the adversary in corrupting P_i in Property 2 of Definitions 9 and 10 (and later also of Definitions 11 and 12). This is to exclude trivial attacks when later analyzing TLS ciphersuites with static Diffie-Hellman key exchange.

5 ACCE Protocols

The notion of (mutually) *authenticated and confidential channel establishment* (ACCE) protocols has just recently been introduced by JKSS [30, 31]. In this section we recall the notion of ACCE, present our modified execution environment and give a definition for server-only ACCE security.

According to JKSS, an ACCE protocol is a protocol executed between two parties which 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. For TLS, the pre-accept phase covers all messages ranging from $Client\ Hello$ to $ServerFinished$ and the phase ends, when both communication partners have established a session key k and reached an accept-state (i.e. $\Lambda = \text{'}\mathtt{accept}\text{'}$). Recall that for TLS $k = k_{\mathsf{enc}}^{\mathsf{Client}}||k_{\mathsf{dec}}^{\mathsf{Client}} = k_{\mathsf{dec}}^{\mathsf{Server}}||k_{\mathsf{enc}}^{\mathsf{Server}}$ as described in Section 3.

POST-ACCEPT PHASE. This phase is entered immediately after the pre-accept phase. In this phase all data is transmitted encrypted under key k with an encryption scheme as given in Section 2.7.

To define security of ACCE protocols, JKSS combined the security model for authenticated key exchange with stateful length-hiding encryption in the sense of [44]. Technically, they provided a slightly modified execution environment that extends the types of queries an adversary may issue. For TLS, the pre-accept phase consists of the TLS Handshake protocol and in the post-accept phase encrypted and authenticated data is transmitted over the TLS Record Layer. As for AKE we will further extend the execution environment to allow for key registration and also give a security definition for server-only authentication.

5.1 Execution Environment

The execution environment is very similar to the model from Section 4, except for a few simple modifications. We again support the case, when the signature scheme used to create certificates is different from the signature scheme implemented in TLS. At first the model is extended such that in the post-accept phase an adversary is also able to 'inject' chosen-plaintexts by making an Encrypt-query[21], 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 and two state variables st_e and st_d for encryption and decryption with a stateful symmetric cipher.

For the post-accept phase each oracle π_i^s keeps additional variables $(u_i^s, v_i^s, C_i^s, \theta_i^s)$. Variables u_i^s, v_i^s are counters, which are initialized to $(u_i^s, v_i^s) := (0,0)$. To simplify our notation in the sequel we additionally define $u_0^0 := 0$. Variable C_i^s is a list of ciphertexts, which initially is empty. We write $C_i^s[u]$ to denote the u-th entry of C_i^s. Variable θ_i^s stores a pair of indices $\theta_i^s \in [\ell] \cup \{0\} \times [d] \cup \{0\}$. If oracle π_i^s accepts having a matching conversation to some other oracle π_j^t, then θ_i^s is set to $\theta_i^s := (j,t)$.[22] Otherwise it is set to $\theta_i^s := (0,0)$.

The key k consists of two different keys $k = (k_{\mathsf{enc}}^{\rho}, k_{\mathsf{dec}}^{\rho})$ for encryption and decryption. Their order depends on the role $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$ of oracle π_i^s. This is the case for TLS.

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 $\Lambda = \text{'}\mathtt{accept}\text{'}$. (Send-queries in an accept-state are handled by the Decrypt-query below).

[21]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 [50] on web servers, or Bard's chosen-plaintext attacks on SSL 3.0 [4, 5].

[22]If there is more than one such oracle, the first in lexicographical order is chosen.

- $\mathsf{Reveal}(\pi_i^s)$, $\mathsf{Corrupt}(P_i)$, and $\mathsf{Register}(pk', aux, proof)$: 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.

$\underline{\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)}$:	$\underline{\mathsf{Decrypt}(\pi_i^s, C, H)}$:
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m_0, st_e)$	$(j,t) := \theta_i^s$
$(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\rho}, \mathsf{len}, H, m_1, st_e)$	$v_i^s := v_i^s + 1$
If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$	If $b_i^s = 0$, then return $\perp$
$u_i^s := u_i^s + 1$	$(m, st_d) = \mathsf{StE.Dec}(k_{\mathsf{dec}}^{\rho}, H, C, st_d)$
$(C_i^s[u_i^s], st_e) := (C^{(b_i^s)}, st_e^{(b_i^s)})$	If $v_i^s > u_j^t$ or $C \neq C_j^t[v_i^s]$, then $\mathsf{phase} := 1$
Return $C_i^s[u_i^s]$	If $\mathsf{phase} = 1$ then return m

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

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

5.2 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 7.

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) and certificates $cert_i$ for all $i \in [\ell]$. The adversary receives the certificates $cert_1, \ldots, cert_\ell$ as input. Now the adversary may start issuing $\mathsf{Send}^{\mathsf{pre}}$, Reveal, Corrupt, Register, Encrypt, and Decrypt queries. Finally, the adversary outputs a triple (i, s, b') and terminates.

MUTUAL AUTHENTICATION We now give our security definition of ACCE protocols that provide mutual authentication, adapted to the modified execution environment.

Definition 11. We say that an adversary (t, ϵ)-*breaks* an ACCE protocol with mutual authentication, 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 $\Pi = j$ such that $id_j \in \{id_1, \ldots, id_\ell\}$, and
   - P_j is τ_j-corrupted with $\tau_0 < \tau_j$, and

- $\mathcal{A}$ did not issue a Reveal-query to oracle π_j^t, such that π_j^t accepted while having a matching conversation to π_i^s (if such an oracle exists), 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 $\Pi = j$ such that $id_j \in \{id_1, \ldots, id_\ell\}$, 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 given by

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

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

We say that an ACCE protocol with mutual authentication is (t, ϵ)-*secure* with perfect forward secrecy, if there exists no adversary that (t, ϵ)-*breaks* it. We say that an ACCE protocol with mutual authentication is (t, ϵ)-*secure* (without perfect forward secrecy), if there exists no adversary that (t, ϵ)-*breaks* it while (in the above) not being allowed to ask the Corrupt query to P_i and P_j (i.e. $\mathcal{P}_i$ and P_j are both τ-corrupted with $\tau = \infty$ when $\mathcal{A}$ terminates).

SERVER-ONLY AUTHENTICATION. We now give our security definition for ACCE protocols, that provide server-only authentication.

Definition 12. We say that an adversary (t, ϵ)-*breaks* an ACCE protocol with server-only authentication (abbreviated as ACCE-SO), 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 that accepts *maliciously* in the sense of Property 1 of Definition 11, and π_i^s has internal state $\rho = \mathsf{Client}$.

2. $\mathcal{A}$ answers the encryption-challenge correctly in the sense of Property 2 of Definition 11 and either

   - oracle π_i^s has internal state $\rho = \mathsf{Client}$, or
   - oracle π_i^s has internal state $\rho = \mathsf{Server}$, and there exists an oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

We say that an ACCE protocol with server-only authentication is (t, ϵ)-*secure* with perfect forward secrecy, if there exists no adversary that (t, ϵ)-*breaks* it. We say that an ACCE protocol with server-only authentication is (t, ϵ)-*secure* (without perfect forward secrecy), if there exists no adversary that (t, ϵ)-*breaks* it while (in the above) not being allowed to ask the Corrupt query to P_i and P_j (i.e. $\mathcal{P}_i$ and P_j are both τ-corrupted with $\tau = \infty$ when $\mathcal{A}$ terminates).

Remark 5. Again, client-only security can be defined analogously, only the internal states ρ of π_i^s and π_j^t are switched. Similar to the AKE, if π_i^s has a matching conversation with π_j^t when only allowing server authentication and π_j^t has a matching conversation with π_i^s when only allowing client authentication then π_i^s and π_j^t have matching conversations and the protocol is ACCE secure. However, the converse is obviously not true in general (see the introduction for a counterexample).

6 TLS-RSA with Server-Only Authentication is ACCE-SO Secure

In this section we show that TLS-RSA with server-only authentication is ACCE-SO secure under Definition 12. Let us first provide the general idea behind all our proofs.

General Proof Idea. For simplicity let us concentrate on server-only authentication. For authentication we consider an adversary that makes the client oracle accept although there is no matching conversation with some other uncorrupted oracle. We consider several cases and subcases. In **Case 1** the adversary does not modify the client Diffie-Hellman share (in TLS-DHE) or the encrypted premaster secret (in TLS-RSA), whereas in **Case 2** it does so. In **Case 1** we now consider two subcases. In **Case 1.1** the adversary does not modify any of the random nonces r_C, r_S. In **Case 1.2** it does make such a modification. In **Case 1.1** we again consider two subcases. In **Case 1.1.1** the adversary does not modify any of the remaining messages m_1 to m_{11} exchanged between client oracle and server oracle, whereas in **Case 1.1.2**, it does so. We now reduce each of these cases to the security of the stateful encryption scheme. We always embed the sLHAE challenge key into $k_S^{(C)}$, the symmetric key that is computed by the client oracle to decrypt the encryption of fin_S. In **Case 1.1.1** and **Case 1.1.2** we have that $k_S^{(C)} = k_S^{(S)}$, i.e. client oracle and server oracle compute the same key for k_S and the queries granted in the sHLAE security game are used to compute the output m_{12} of the server oracle.

We have to show that the adversary *even with the help of the server oracle* cannot make the client oracle accept without breaking one of the underlying security assumptions.

In **Case 1.1.1**, the master secrets, the encryption keys and the fin_S messages computed by client and server oracle are equal and all indistinguishable from random from the adversary's point of view. By the definition of security, the adversary must thus have computed a *new* encryption m'_{12} of fin_S which is distinct from the message m_{12} that is computed by the server oracle to make the client oracle accept. If we embed the sLHAE challenge key into $k_S^{(C)} = k_S^{(S)}$, and use the encryption queries granted in the sLHAE security game to compute m_{12} we can directly break the security of the sLHAE scheme. In **Case 1.1.2** the master secret and encryption keys computed by client and server oracle are equal and indistinguishable from random. However, when client and server oracle compute the Finished messages they use distinct transcripts. First, by the security of the hash function the hash values of these transcripts differ as well. Second, by the security of the PRF and since the input values to the PRF are distinct, the values for fin_S computed by the client oracle and server oracle, $fin_S^{(C)}$ and $fin_S^{(S)}$, must differ as well for the two oracles with overwhelming probability $2^{-\mu}$. In this case the adversary cannot use the output of the server oracle to make the client oracle accept because due to the correctness of the sLHAE encryption system there cannot be two distinct plaintext for one ciphertext. Instead it has to compute a new encryption of $fin_S^{(C)}$ on its own. Such an adversary can directly be used to break the security of the sLHAE scheme.

In **Case 1.2** the adversary modifies (at least) one of the nonces r_C, r_S exchanged between client and server oracle. In **Case 2** we have that the adversary modifies the message m_8 sent by the client, such that the premaster secrets computed by client and server are distinct with overwhelming probability. The proofs for **Case 1.2** and **Case 2** are similar. In **Case 1.2** we first have to show for each ciphersuite that the premaster secret pms exchanged between client and server oracle is indistinguishable from a random value to the adversary. In **Case 2**, we need to show that $pms^{(C)}$ is indistinguishable from a random value to the adversary and that it is distinct from (and actually independent of) $pms^{(S)}$ (with probability 2^{ν} where ν is the bitsize of pms) because of the adversary's modifications to m_8. Next we have to show that the master secret computed by the client oracle $ms^{(C)}$ is indistinguishable from random and distinct to $ms^{(S)}$. In **Case 2** this is because the key to the PRF, $pms^{(C)}$, is already indistinguishable from random and distinct from

$pms^{(S)}$. In **Case 1.2**, this is because client and server use distinct nonces r_S, r_C as input to the PRF to compute the master secrets while $pms^{(C)} = pms^{(S)}$ is indistinguishable from random to the adversary. So by the security of the PRF, with probability $2^{-\mu}$ $ms^{(C)}$ is distinct from $ms^{(S)}$. We can now substitute the output values $k_C^{(C)}, k_S^{(C)}, fin_C^{(C)}, fin_S^{(C)}$ of the second application of the PRF with truly random values. By the security of the PRF the adversary cannot recognize this modification. Now, since a) $k_S^{(C)}$ is drawn uniformly at random and b) $k_S^{(C)}$ is never used at any point before, we can draw $k_S^{(C)}$ also after the client oracle has received the last protocol message m_{12} (and independently of the adversary and all the computations within the server oracle). This shows that no information about the random key $k_S^{(C)}$ is leaked. We can thus embed the sLHAE challenge key in $k_S^{(C)}$. If the adversary makes the client oracle accept we can directly break the security of the sLHAE scheme. In **Case 1.2** and **Case 2**, we do not have to exploit any of the Encrypt queries granted by the sLHAE security game.

This shows authentication. To show that the adversary cannot break the encryption challenge we observe that *in TLS the client only accepts, if all the values used for the computation of the encryption keys are unmodified*. In particular, **Case 1.1.2** also rules out that the adversary modifies one of the remaining messages which are sent from the client oracle to the server oracle. If the client oracle accepts, both client and server oracle thus have computed the same encryption keys. We can now directly plugin the sLHAE challenge either in $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. Any adversary that breaks the encryption challenge can be used to break the sLHAE security.

Theorem 1. *Let $\mu = \mu(\kappa)$ be the output length of the PRF, $\nu = \nu(\kappa)$ be the length of the premaster secret, and $\lambda = \lambda(\kappa)$ be the length of the nonces r_C and r_S. Assume that the pseudo-random function is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the public key encryption scheme is $(t, \epsilon_{\mathsf{pke}}, q_{\mathsf{pke}})$-secure, the signature scheme used for generating certificates is $(t, \epsilon_{\mathsf{ca}}, q_{\mathsf{ca}})$-secure, and the sLHAE scheme is $(t, \epsilon_{\mathsf{slhae}})$-secure. Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the server-only authenticated TLS-RSA Handshake protocol in the sense of Definition 11 with $t \approx t'$ and $q_{\mathsf{pke}} \geq d$, $q_{\mathsf{prf}} \geq 2$, and $q_{\mathsf{ca}} \geq w + \ell$[23] it holds that*

$$\epsilon_{\mathsf{tls}} \leq 2\epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-1}} + 8d\ell^2(2^{-\nu} + 2^{-\mu} + \epsilon_{\mathsf{H}}) + (8d\ell^2 + d\ell)(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{slhae}}).$$

We prove Theorem 1 via two lemmas. Lemma 1 bounds the probability ϵ_{auth} that an adversary succeeds in making the client oracle accept maliciously. Lemma 2 bounds the probability ϵ_{enc} that an adversary answers the encryption challenge correctly while not making client oracle accept maliciously. Then we have

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

6.1 Server-Only Authentication

Lemma 1. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists a client oracle π_i^s that accepts maliciously with respect to Definition 12 is at most*

$$\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda}} + 4d\ell^2(\epsilon_{\mathsf{slhae}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + 2^{-\nu} + 2^{-\mu} + \epsilon_{\mathsf{pke}}).$$

where all quantities are defined as stated in Theorem 1.

[23]Meaning that the total number of issued certificates is smaller than q_{ca}.

PROOF. The proof proceeds in a *sequence of games*, following [9, 48]. 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. In particular, the difference of the success probability of the adversary in two subsequent game is negligible. 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 a client oracle accepts maliciously in the sense of Definition 12 in Game δ. If required, we use $x^{(S)}$ to denote the value of x that is computed by a server oracle and $x^{(C)}$ the value of x that is computed by a client oracle (e.g. $pms^{(S)}$ denotes the premaster secret computed by the server).

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

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

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_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 $(d\ell)^2 \cdot 2^{-\lambda}$, which implies

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

Note that now each oracle has a unique nonce r_C or r_S, which will be used to compute the master secret.

Game 2. The challenge proceeds as before, but we now abort if the adversary forges a certificate that binds an $id_i \in \{id_1, \ldots, id_\ell\}$ to a new public key. It is easy to see that the probability of this to happen is bounded by the probability to break the signature scheme since we can use such an adversary to break the security of the signature game ϵ_{sig}. The challenger behaves exactly as before except that it embeds the signature challenge pk^* in the key used for certification $pk := pk^*$. The signature queries granted in the signature security game are used to issue all the certificates on the identities $id_1, \ldots, id_\ell$ and to answer all $\mathsf{Register}$ queries. The challenger answers all other queries exactly as before. In this way any adversary that forges a certificate can directly be used to break the security of the signature scheme. Clearly we have

$$\Pr[\mathsf{break}_1^{(1)}] \leq \Pr[\mathsf{break}_2^{(1)}] + \epsilon_{\mathsf{ca}}.$$

Game 3. We try to guess which client oracle will be the first to accept maliciously. If our guess is wrong, we abort the game.

Technically, this game is identical to the previous, except for the following. The challenger guesses two random indices $(i^*, s^*) \xleftarrow{\$} [\ell] \times [d]$. If π_i^s is the first client oracle that 'accepts' maliciously and $(i,s) \neq (i^*, s^*)$, then the challenger aborts the game. Since there are at most $d\ell$ oracles the probability to guess correctly, i.e. we have $(i,s) = (i^*, s^*)$, is $\geq 1/(d\ell)$, and thus

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

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

Game 4. In this game we guess the party that holds the communication partner of the client oracle. Technically we draw $j^* \xleftarrow{\$} [\ell]$. If $\Pi \neq j^*$ for client oracle $\pi_{i^*}^{s^*}$ we abort.

We get

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

In the following we will consider two distinct cases. In the first case, **Case 1**, the adversary does not modify the ciphertext containing the premaster secret, message m_8, which is sent from the client oracle to the server oracle. In the second case, **Case 2**, it does so.

<u>**Case 1:**</u>

Game 5. In this game we exchange the ciphertext c of the premaster secret sent by $\pi_{i^*}^{s^*}$, message m_8, by an encryption c' of a truly random message r (and independently of pms). However, both client and server continue to use pms as computed by the client. Since the challenger implements all server oracles it can, whenever the ciphertext c' is received by any server oracle of P_{j^*}, set the premaster secret to pms. We show that this modification is indistinguishable from the previous game when the PKE is secure. Any adversary that can distinguish these two games can be used to break the security of the public key scheme as follows.

First, we embed the challenge public key of the PKE challenger in pk_{j^*}. Next we draw a random pms for oracle $\pi_{i^*}^{s^*}$ and send it to the PKE challenger. The challenge in turn responds with an encryption c^* of either pms or an independently drawn random value r. Let $\pi_{j^*}^{t}$ be the server oracle where we exchange m_8 with c^* but continue to use the original pms. For all other oracles $\pi_{j^*}^{t}$ of P_{j^*} with $t \in [d]$ and $t \neq t^*$ we use the decryption queries granted in the PKE security game to decrypt m_8 and set the pms of the server oracle as given in the plaintext of m_8 unless the server oracle receives c^*. Whenever c^* is received by any server oracle we simply set the premaster secret of that oracle to pms. Observe that if c^* is an encryption of pms we are in the previous game. However, if c^* encrypts an independently drawn random message we are in the current game. So any attacker that distinguishes these two games can directly be used to break the security of the PKE scheme.

We have

$$\Pr[\mathsf{break}_4^{(1)}] \leq \Pr[\mathsf{break}_5^{(1)}] + \epsilon_{\mathsf{pke}}.$$

We note that in this game the adversary does not obtain any information on pms from m_8.

We now consider two subcases. Either the adversary does not modify any of the messages r_C or r_S in transit (**Case 1.1**) or it does so (**Case 1.2**).

<u>**Case 1.1:**</u>

Game 6. In this game, we substitute the master secret generated by the client oracle and the server oracle by a single truly random value. Any adversary that can distinguish between this and the previous game can be used to break the security of the function PRF.

To show this, observe that since pms is drawn uniformly at random and no information about pms is revealed to the attacker via m_8 due the previous game we can directly embed the security challenge of the PRF challenger: we use a PRF query (granted in the PRF security game) to compute the master secret. If the answer has been computed with PRF we are in the previous game. If the answer is a truly random value,

we are in the current game. So any attacker that distinguishes this game from the previous can directly be used to break the security of the PRF.

We have,

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

Recall that neither r_S nor r_C have been modified by the adversary. We are now in a game where the master secret ms of client and server oracle are equal and the truly random function used for computing ms can only be accessed by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$.
At this point we define another two subcases. Either the adversary does not modified any of the (remaining) messages of m_1 to m_{11}, **Case 1.1.1**, or it does, **Case 1.1.2**.

Case 1.1.1:

In this case, the adversary must by definition modify m_{12} to make the client oracle accept maliciously.

Game 7. However, we now guarantee, that the adversary has no knowledge of the encryption keys generated by the client oracle and the server oracle. We do so by first replacing the PRF used to compute these keys (and fin_S) by a truly random function, and then substituting fin_S and the encryption keys k_S and k_C with truly random values. Any adversary that can detect this modification can be used to break the security of the PRF.

To show this, observe that since ms is drawn uniformly at random we can directly embed the security challenge of the PRF challenger: we use a PRF query (granted in the PRF security game) to compute the encryption keys and fin_S. If these values have been computed with the PRF, we are in the previous game. If these values are truly random, we are in the current game. So any attacker that distinguishes this game from the previous can directly be used to break the security of the PRF.

We have

$$\Pr[\mathsf{break}_6^{(1)}] \leq \Pr[\mathsf{break}_7^{(1)}] + \epsilon_{\mathsf{prf}}.$$

Game 8. In this game we show that any successfull adversary can be used to break the security of the encryption system. We know that the adversary outputs a message $m'_{12} \neq m_{12}$ which makes the client adversary accept. This can only happen if m'_{12} is a distinct, valid encryption of m_{12}. However, then m'_{12} can be used to break the security of the sLHAE encryption scheme as follows: since the encryption keys are random we can directly plug-in the sLHAE challenge (in k_S) that is used to encrypt and decrypt messages sent from the server oracle to the client oracle. When we need to generate m_{12} that is computed by the server oracle we use one $\mathsf{Encrypt}$ query as granted by the sLHAE challenger. Since the adversary outputs a new ciphertext m'_{12} that has not been generated by $\mathsf{Encrypt}$, this directly breaks the security of the sLHAE game.

$$\Pr[\mathsf{break}_7^{(1)}] \leq \epsilon_{\mathsf{slhae}}.$$

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + d\ell^2(\epsilon_{\mathsf{slhae}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{pke}}).$$

In the following, we will only explain the differences to the proof of **Case 1.1.1**.

Case 1.1.2:

In this case, the adversary has modified any of the remaining messages m_1 to m_{11} on transit. We now substitute the output values of the pseudo-random function keyed with ms by truly random values. Since the encryption keys do only depend on r_C and r_S, we can substitute them by the same random keys. However, the computation of fin_S does also depend on the messages m_1 to m_{11}. By assumption, the values m_1 to m_{11} computed and received by the two oracles do differ at some point. Thus, when hashing the concatenated messages, the corresponding outputs can only be equal with probability at most ϵ_{H} – otherwise we can use the adversary to produce a collision and break a security of the hash function. Since the inputs to PRF are now distinct with overwhelming probability we can substitute $\mathit{fin}_S^{(S)}$ and $\mathit{fin}_S^{(C)}$[24] by independently drawn random values. With the same argument as before, our modification can only be detected by the adversary with probability ϵ_{prf}. We now abort if the output values of the PRF are equal which only happens with probability $2^{-\mu}$.

At this point the server oracle cannot help the adversary to compute a message m_{12} which will make the client oracle accept. Thus, to make the client oracle accept, the adversary must compute m'_{12} to be an encryption of $\mathit{fin}_S^{(C)}$. However, since $\mathit{fin}_S^{(S)}$ is distinct from $\mathit{fin}_S^{(C)}$ the adversary cannot use m_{12}.

We now embed the sLHAE challenge in $k_S^{(C)}$ which is used by the client to decrypt messages sent by the server. Then the adversary against server-only authentication outputs a new ciphertext m'_{12} that has not been generated by Encrypt which breaks the security of the sLHAE game.

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \le \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + d\ell^2(\epsilon_{\mathsf{slhae}} + 2\epsilon_{\mathsf{prf}} + 2^{-\mu} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{pke}}).$$

Case 1.2:

In this case the adversary has modified either r_C or r_S in transit. Since these values are input to the PRF when computing the master secret we can substitute $ms^{(S)}$ and $ms^{(C)}$ by independently drawn random values. We abort if these values are equal which only happens with probability $2^{-\mu}$. In the next step we replace $\mathit{fin}_C^{(C)}$ and $\mathit{fin}_S^{(C)}$ and the encryption keys $k_S^{(C)}$ $k_C^{(C)}$ with independently drawn truly random values. Due to the security of the PRF these modifications cannot be detected by the adversary. Since $k_S^{(C)}$ is never used at any point before we can also draw this value after the client oracle receives the encryption of fin_S and independently of all the computations of the adversary and the server oracle. We now embed the sLHAE challenge into $k_S^{(C)}$.

To make the client oracle accept the adversary must compute m'_{12} to be an encryption of $\mathit{fin}_S^{(C)}$ under $k_S^{(C)}$. Thus, if the adversary outputs a new ciphertext m'_{12}, it directly breaks the security of the sLHAE game as before. Observe that to simulate the server oracle correctly, we do not have to query Encrypt to generate m_{12}.

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \le \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + d\ell^2(\epsilon_{\mathsf{slhae}} + 2\epsilon_{\mathsf{prf}} + 2^{-\mu} + \epsilon_{\mathsf{pke}}).$$

[24]Recall that $\mathit{fin}_S^{(C)}$ refers to the Server Finished message that is re-computed by the client (for verification)

Case 2:

We now have that the adversary modifies message m_8 which is sent from the client oracle to the server oracle and contains the premaster secret pms. We first substitute the ciphertext c (i.e. the encrypted premaster secret) that is sent by the client oracle, by an encryption of a truly random message r independently drawn from pms. If r is equal to pms which happens with negligible probability $2^{-\nu}$ we abort, otherwise we continue. The client will continue to use the randomly drawn pms. The server oracle will use whatever it receives as m_8. We show that our modification is indistinguishable from the previous game when the PKE is secure. Any adversary that can distinguish these two games can be used to break the security of the public key scheme as follows.

We embed the challenge public key of the PKE challenger in pk_{j^*}. For all other oracles $\pi_{j^*}^t$ of P_{j^*} with $t \in [d]$ and $t \neq t^*$ we use the decryption queries granted by the PKE challenger to decrypt m_8 messages. We let the client oracle draw the random pms and then sent it to the PKE challenger who returns a ciphertext c^* which either encrypts pms or a truly random value. Next we send c^* to the server oracle $\pi_{j^*}^{t^*}$. Observe that if c^* is an encryption of pms we are in the previous game. However, if c^* encrypts an independently drawn random message we are in the current game. So any attacker that distinguishes these two games can directly be used to break the security of the PKE scheme.

We now have that $pms^{(C)}$ and $pms^{(S)}$ are independent (while $pms^{(C)}$ is uniformly random). Observe that no information about $pms^{(C)}$ is revealed neither to the adversary nor to the server oracle. In the next step we replace the master secret $ms^{(C)}$ by a truly random value. Then with the same arguments as above we replace the Finished messages $fin_C^{(C)}$ and $fin_S^{(C)}$ and the encryption keys $k_S^{(C)}$ $k_C^{(C)}$ by independently drawn truly random values. These modifications are all indistinguishable by the adversary by the security of the PRF. Observe that now $k_S^{(C)}$ can also be drawn after the client oracle receives m_{12} (and independently of all previous values) since it is never used before. Now we can embed the sLHAE challenge key into $k_S^{(C)}$ and any message that makes the client oracle accept can be used to break the sLHAE scheme. We do not have to use any Encrypt query granted by the sLHAE security game.

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + d\ell^2(\epsilon_{\mathsf{slhae}} + 2\epsilon_{\mathsf{prf}} + 2^{-\nu} + \epsilon_{\mathsf{pke}}).$$

□

6.2 Indistinguishability of Keys

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

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + d\ell(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{slhae}}).$$

where ϵ_{auth} is an upper bound on the probability that there exists a client oracle that accepts maliciously in the sense of Definition 12 (Lemmas 1) and all other quantities are defined as stated in Theorem 1.

PROOF. Assume without loss of generality that the adversary $\mathcal{A}$ always outputs a triple such that all conditions in Property 2 of Definition 12 are satisfied. Let $\mathsf{break}_\delta^{(2)}$ denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by the client oracle π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does

not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(2)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ.

Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment described in Section 5. For some ϵ_{enc} we have

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

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 the ACCE definition.

Thus we have

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

where ϵ_{auth} is an upper bound on the probability that there exists a client oracle that accepts maliciously in the sense of Definition 12 (cf. Lemma 1).

Recall that we now assume that $\mathcal{A}$ outputs a triple (i, s, b') such that the oracle π_i^s 'accepts' with 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 $(i, s) \neq (i^*, s^*)$. With probability $1/(d\ell)$ we have $(i, s) = (i^*, s^*)$, 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 the previous game. 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 $\pi_{i^*}^{s^*}$.

Subsequently, we only consider the case that the adversary does not modify any of the messages of the pre-accept phase.

Due to proof of Lemma 1, we can subsequently only deal with adversaries that do not modify messages exchanged between client $\pi_{i^*}^{s^*}$ and server oracle. Then we can use all the game-hops of the previous proof to the position where both the server and the client oracle compute the keys used for the sLHAE scheme as uniformly random keys. In this way we end up in Game 7 of the previous proof.

Game 3. In this game we can directly plug-in the sLHAE challenge. We randomly decide to embed the sLHAE challenge key into $k_C^{(C)} = k_C^{(S)}$ or into $k_S^{(C)} = k_S^{(S)}$. With probability $\geq 1/2$ our choice is correct (i.e. the adversary attacks the sLHAE ciphertexts generated under this key). We have

$$\mathsf{Adv}_2 \leq 2\ell \cdot \mathsf{Adv}_3.$$

This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the $\mathsf{Encrypt}$ query of the sLHAE game. Similarly the $\mathsf{Decrypt}$ query is used to decrypt all the queries on behalf of the receiving oracle. If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then the challenger forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the challenger essentially relays all

messages it is easy to see that an attacker $\mathcal{A}$ having advantage ϵ' yields an attacker 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

$$\mathsf{Adv}_3 \leq 1/2 + \epsilon_{\mathsf{slhae}}.$$

Collecting probabilities we get that $\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + d\ell(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{slhae}})$.

□

7 TLS-DH with Mutual Authentication is ACCE Secure

In this section we show that TLS-SDH with mutual authentication is ACCE secure under Definition 11. We would like to briefly comment on why it is seemingly not possible to rely on (reasonable) variants of the PRF-ODH assumption in the security proof. Assume the client oracle uses shares g^u and g^v. We basically need that $\mathsf{PRF}(g^{uv}, m^*)$ for some m^* must be indistinguishable from random. Thus, the proof requires that u *and* v are hidden to the simulator, otherwise we could trivially break the assumption. Also, client and server may communicate in several sessions with $m \neq m^*$. So, the assumption not only has to help us to compute the master secret when one of the shares is g^u or g^v (this could be handled by PRF-ODH-like assumptions alone) but also for distinct messages. However, we also need to simulate communications with adversarially generated parties, i.e. where the simulator may not know the corresponding secret key. Recall the challenger of the PRF-ODH assumption (for g^u) computes $\mathsf{PRF}(h^u, m)$ when given message m and another group element h. One could try to use $h = g^v$. However then the assumption would not allow to simulate communications for g^v with parties generated by the adversary. One would thus have to use two PRF-ODH challenger which are strongly related to each other. We do not feel such an assumption is reasonable and instead rely on the KOSK assumption in the security proof.

Theorem 2. *Let μ be the output length of the PRF and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function is $(t, \epsilon_{\mathsf{prf}}, qprf)$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the signature scheme used for generating certificates is $(t, \epsilon_{\mathsf{ca}}, q_{\mathsf{ca}})$-secure, the DDH-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, and the sLHAE scheme is $(t, \epsilon_{\mathsf{slhae}})$-secure. Assume that no party will accept in a TLS handshake, if the public key of the communication partner is equal to the public key of that party.*[25]

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the static Diffie-Hellman TLS Handshake protocol with mutual authentication in the sense of Definition 11 with $t \approx t'$ and $q_{\mathsf{prf}} \geq 2$, $q_{\mathsf{prfodh}} \geq d$, and $q_{\mathsf{ca}} \geq w + \ell$ it holds that

$$\begin{aligned} \epsilon_{\mathsf{tls}} &\leq 4\epsilon_{\mathsf{ca}} + 4\frac{(d\ell)^2}{2^\lambda} + 4\ell^2 \cdot \epsilon_{\mathsf{ddh}} + 12d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu}) + \ell^2(\epsilon_{\mathsf{ddh}} + d(2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{slhae}})) \\ &\leq 4\epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-2}} + 5\ell^2 \cdot \epsilon_{\mathsf{ddh}} + 26d\ell^2 \cdot \epsilon_{\mathsf{prf}} + 14d\ell^2 \cdot \epsilon_{\mathsf{slhae}} + 12d\ell^2 \cdot \epsilon_{\mathsf{H}}. \end{aligned}$$

We prove Theorem 2 by proving two lemmas. Lemma 3 bounds the probability ϵ_{auth} that an adversary succeeds in making any oracle accept maliciously. Lemma 6 bounds the probability ϵ_{enc} that an adversary does not succeed in making an oracle accept maliciously, but which answers the encryption challenge correctly. Then we have

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

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

$$\epsilon_{\mathsf{auth}} \leq 2\epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-1}} + 2\ell^2 \cdot \epsilon_{\mathsf{ddh}} + 6d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu}),$$

where all quantities are defined as stated in Theorem 2.

[25] This can easily be realized technically.

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 11, and $\epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Server}$ that accepts maliciously. We claim that

$$\epsilon_{\mathsf{client}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 3d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu}), \text{ and}$$
$$\epsilon_{\mathsf{server}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 3d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu}),$$

and thus

$$\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} \leq 2 \cdot \left(\epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 3d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu})\right).$$

We split up the proof of Lemma 3 in two separate lemmas, Lemma 4 and Lemma 5, which we give (and prove) in the following sections.

7.1 Client Authentication

Lemma 4. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists any π_i^s with $\rho = \mathsf{Client}$ that accepts maliciously with respect to Definition 11 is at most Collecting probabilities we get that*

$$\epsilon_{\mathsf{client}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 3d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu})$$

where all quantities are defined as stated in Theorem 2.

PROOF. The proof again proceeds in a *sequence of games*. As before we use different subcases to organize the proof. When denoting the different subcases we stay consistent with the general proof outline. We end up in a final game, where no adversary can break the security of the protocol. Let $\mathsf{break}_\delta^{(3)}$ be the event that occurs when a client oracle accepts maliciously in the sense of Definition 11 in Game δ.

Game 0. This game equals the ACCE security experiment described in Section 5. Thus, for some $\epsilon_{\mathsf{client}}$ we have

$$\Pr[\mathsf{break}_0^{(3)}] = \epsilon_{\mathsf{client}}.$$

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

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

Note that now each oracle has a unique nonce r_C or r_S, which will be used to compute the master secret.

Game 2. The challenge proceeds as before, but we now abort if the adversary forges a certificate that binds an $id_i \in \{id_1, \ldots, id_\ell\}$ to a new public key. It is easy to see that the probability of this to happen is bounded by the probability to break the signature scheme since we can use such an adversary to break the security of the signature game ϵ_{sig}. The challenger behaves exactly as before except that it embeds the signature challenge pk^* in $pk := pk^*$. The signature queries granted by the signature game are used to issue all the certificates on the identities $id_1, \ldots, id_\ell$ and to answer all Register queries. The challenger answers all other queries exactly as before. In this way any adversary that forges a certificate can directly be used to break the security of the signature scheme.

Clearly we have

$$\Pr[\mathsf{break}_1^{(3)}] \leq \Pr[\mathsf{break}_2^{(3)}] + \epsilon_{\mathsf{ca}}.$$

Game 3. We now try to guess two distinct indices (i^*, j^*) with $i^*, j^* \in [\ell]$, such that the first oracle π_i^s that will accept maliciously with $\rho = \mathsf{Client}$ belongs to party P_{i^*}, i.e. $i = i^*$ and has communication partner $\Pi = j$ with $j = j^*$. If the first oracle to accept maliciously is π_i^s with $i \neq i^*$ or π_i^s has $\Pi = j$ with $j \neq j^*$, then the challenger aborts the game. Since there are at most ℓ parties, the probability to guess correctly, i.e. we have $(i, j) = (i^*, j^*)$, is $\geq 1/\ell^2$, and thus

$$\Pr[\mathsf{break}_2^{(3)}] \leq \ell^2 \cdot \Pr[\mathsf{break}_3^{(3)}].$$

Game 4. In this game we want to substitute the premaster secret pms computed by all oracles $\pi_{i^*}^s$ and their partner oracles $\pi_{j^*}^t$ for $s, t \in [d]$ by a randomly chosen value. We can exploit the fact that the premaster secret will be the same for all sessions of parties P_{i^*} and P_{j^*} in this setting (static Diffie-Hellman keys and mutual authentication) and that we know the party P_{i^*} holding the oracle that will accept maliciously first and its communication partner P_{j^*} due to the previous game.

The challenger in this game proceeds as before but replaces the Diffie-Hellman public keys in the certificates for P_{i^*} and P_{j^*} with the Diffie Hellman values (g^a, g^b) from a DDH challenge tuple (g, g^a, g^b, g^c). Observe that g^a and g^b are distributed *exactly* as the public keys that have originally been chosen by the challenger. The challenger then proceeds as follows. Whenever two oracles of P_{i^*} and P_{j^*} communicate with each other, the premaster secret will be set to g^c, otherwise the pms will be computed honestly and then set by the challenger. The challenger can do so, since we know the secret keys for all public keys due to the KOSK assumption. It knows the corresponding secret Diffie-Hellman key of the communication partner of P_{i^*} or P_{j^*} and can so compute the premaster secret.

If $g^c = g^{ab}$ then we are in the previous game, whereas if $g^c \neq g^{ab}$ we are in the current game. It follows that we can use an adversary that can distinguish between this game and the previous game to break DDH.

We have

$$\Pr[\mathsf{break}_3^{(3)}] \leq \Pr[\mathsf{break}_4^{(3)}] + \epsilon_{\mathsf{ddh}}.$$

Game 5. We now try to guess index s^* with $s^* \in [d]$, such that the first oracle of P_{i^*} that will accept maliciously is $\pi_{i^*}^{s^*}$. If the first oracle to accept maliciously is $\pi_{i^*}^s$ with $s \neq s^*$, then the challenger aborts the game. Since there are at most d oracles for each party, the probability to guess correctly, i.e. we have $s = s^*$, is $\geq 1/d$, and thus

$$\Pr[\mathsf{break}_4^{(3)}] \leq d \cdot \Pr[\mathsf{break}_5^{(3)}].$$

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

We now consider two cases. Either the adversary does not modify any of the messages r_C or r_S in transit (**Case 1.1**) or it does so (**Case 1.2**).

<u>**Case 1.1:**</u>

Game 6. In this game, we substitute the master secret generated by the client oracle and the server oracle by a truly random value. Any adversary that can distinguish between this and the previous game can be used to break the security of the PRF.

To show this, observe that since pms is drawn uniformly at random due to the previous game, we can directly embed the security challenge of the PRF challenger: we use a PRF query to compute the master secret. If the answer has been computed with the PRF we are in the previous game. If the answer is a truly random value, we are in the current game. So any attacker that distinguishes this game from the previous can directly be used to break the security of the PRF.

We have,

$$\Pr[\mathsf{break}_5^{(3)}] \leq \Pr[\mathsf{break}_6^{(3)}] + \epsilon_{\mathsf{prf}}.$$

We are now in a game where the master secret of client and server oracle are equal and truly random, and where neither r_S nor r_C have been modified by the adversary. At this point we define two subcases. Either the adversary does not modify any of the (remaining) messages m_1 to m_{11}, **Case 1.1.1**, or it does, **Case 1.1.2**.

<u>**Case 1.1.1:**</u>

Game 7. In this game, the adversary must by definition modify m_{12} to make the client oracle accept maliciously. However, we first substitute fin_S and the encryption keys generated by the client oracle and the server oracle by truly random values. Any adversary that can distinguish between this and the previous game can be used to break the security of the PRF. To show this, observe that since ms is drawn uniformly at random we can directly embed the security challenge of the PRF challenger: we use a PRF query to compute fin_S. If the answer has been computed with the PRF we are in the previous game. If the answer is a truly random value, we are in the current game. So any attacker that distinguishes this game from the previous can directly be used to break the security of the PRF. We have,

$$\Pr[\mathsf{break}_6^{(3)}] \leq \Pr[\mathsf{break}_7^{(3)}] + \epsilon_{\mathsf{prf}}.$$

Game 8. In this game we show that any adversary that wins can be used to break the security of the encryption system. We know that the adversary outputs a message $m'_{12} \neq m_{12}$ which makes the client adversary accept. Since fin_S is a truly random value this can only happen if m'_{12} is also a valid encryption of m_{12}. However, this encryption can be used to break the security of the sLHAE encryption scheme as follows: we embed the sLHAE challenge key into $k_S^{(C)} = k_S^{(S)}$ and use the $\mathsf{Encrypt}$ query to compute m_{12}. Now the adversary outputs a new ciphertext m'_{12} that has not been generated by $\mathsf{Encrypt}$ which breaks the security of the sLHAE game.

$$\Pr[\mathsf{break}_7^{(3)}] \leq \epsilon_{\mathsf{slhae}}.$$

Collecting probabilities we get that

$$\epsilon_{\mathsf{client}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{slhae}}).$$

In the following, we will only explain the differences to the proof of **Case 1.1.1**.

Case 1.1.2:

In this case, the adversary has modified any of the remaining messages m_1 to m_{11} on transit. We first substitute the output values of the PRF when keyed with ms by truly random values. Since the encryption keys do only depend on r_C and r_S we substitute the values that have been computed by the client oracle and the server oracle by the same random keys. However, the computation of fin_S does depend on the messages m_1 to m_{11}. Since m_1 to m_{11} are distinct, so must be their hash values, as otherwise we can use the adversary to break the security of the hash function. The distinct hash values are input to the PRF and we can substitute $\mathit{fin}_S^{(S)}$, $\mathit{fin}_S^{(C)}$ and $k_S^{(C)} = k_S^{(S)}$ by independently drawn random values. If $\mathit{fin}_S^{(C)} = \mathit{fin}_S^{(S)}$, which happens with probability $2^{-\mu}$, we abort. Otherwise ($\mathit{fin}_S^{(C)} \neq \mathit{fin}_S^{(S)}$), the server oracle will not compute a message m_{12} that will make the client oracle accept. To make the client oracle accept the adversary must compute m'_{12} to be an encryption of $\mathit{fin}_S^{(C)}$.

Now we again embed the sLHAE challenge in $k_S^{(C)} = k_S^{(S)}$. Then the adversary outputs a new ciphertext m'_{12} that has not been generated by $\mathsf{Encrypt}$ which breaks the security of the sLHAE game.

Collecting probabilities we get that

$$\epsilon_{\mathsf{client}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu}).$$

Case 1.2:

In this case the adversary has modified either r_C or r_S in transit. Since these values are input to the PRF when computing the master secret we can by the security of the PRF substitute $ms^{(S)}$ and $ms^{(C)}$ by independently drawn random values. With probability $2^{-\mu}$ these values are equal and we abort. Otherwise we proceed as follows. Similar to before, we can by the security of the PRF show that $k_S^{(C)}$ can be substituted with a uniformly random key that is drawn after the client oracle receives the last message m_{12}. Thus $k_S^{(C)}$ is independent of the server oracle and the adversary. Since $k_S^{(C)}$ is a uniformly random value independent of any value computed by the server oracle and unknown to the adversary we can as before embed the PRF challenge in $k_S^{(C)}$.

Thus, if the adversary outputs a message that makes the client accept, it directly breaks the security of the sLHAE game.

Collecting probabilities we get that

$$\epsilon_{\mathsf{client}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu}).$$

□

7.2 Server Authentication

Lemma 5. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists a π_i^s with $\rho = \mathsf{Server}$ that accepts maliciously with respect to Definition 11 is at most*

$$\epsilon_{\mathsf{server}} \leq \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^\lambda} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 3d\ell^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{slhae}} + 2^{-\mu})$$

where all quantities are defined as stated in Theorem 2.

PROOF. Due to the symmetry of the handshake the proof is essentially that of Lemma 4. The only difference is that now the message m_{12} can be modified in 'transit'. However by the definition of matching conversation this does not result in a successfull adversary. □

7.3 Indistinguishability of Keys

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

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 2d\ell^2 \cdot (\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{slhae}}).$$

where ϵ_{auth} is an upper bound on the probability that there exists a client oracle that accepts maliciously in the sense of Definition 11 (Lemmas 4) and all other quantities are defined as stated in Theorem 2.

PROOF. Assume without loss of generality that the $\mathcal{A}$ always outputs a triple such that all conditions in Property 2 of Definition 11 are satisfied. Let $\mathsf{break}_\delta^{(5)}$ denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by the oracle π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(5)}] - 1/2$ 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 5. For some ϵ_{enc} we have

$$\Pr[\mathsf{break}_0^{(4)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0.$$

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 the ACCE definition.

Thus we have

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

where ϵ_{auth} an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 11 (cf. Lemma 4).

Recall that we now assume that $\mathcal{A}$ outputs a triple (i, s, b') such that the oracle π_i^s 'accepts' with 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. We now try to guess indices (i^*, j^*) with $i^*, j^* \in [\ell]$, such that when the Adversary terminates and outputs (i, s, b') it holds that $i = i^*$ and oracle $\pi^s_{i^*}$ has communication partner $\Pi = j$ with $j = j^*$.

If the Adversary terminates and outputs (i, s, b'), and $i \neq i^*$ or π^s_i has communication partner $\Pi = j$ with $j \neq j^*$, the challenger aborts the game. Since there are at most ℓ parties, the probability to guess correctly, i.e. we have $(i, j) = (i^*, j^*)$, is $\geq 1/\ell^2$, and thus

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

Game 3. With the same arguments as in Game 4 of the proof of Lemma 4 we exchange the premaster secret pms computed in all sessions between parties P_{i^*} and P_{j^*} with a randomly chosen value.

Thus we have

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

Game 4. We now try to guess index s^* with $s^* \in [d]$, such that when the Adversary terminates and outputs (i, s, b') it holds that $s = s^*$. Since there are at most d oracles for each party, the probability to guess correctly, i.e. we have $s = s^*$, is $\geq 1/d$, and thus

$$\mathsf{Adv}_3 \leq d \cdot \mathsf{Adv}_4.$$

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

Game 5. In this game we exchange the encryption keys computed by $\pi^{s^*}_{i^*}$ (and the partner oracle having a matching conversation with π^s_i) with uniformly random keys. To show this, we, for reasons of simplicity, will make several changes at the same time and argue for each, that the adversary cannot detect this change.

MASTER SECRET. With the same arguments as in the previous proof we exchange the master secret with a uniformly random value.

ENCRYPTION KEYS. With the same arguments as in the previous proof we exchange the encryption keys with a uniformly random value. Observe that since ms is drawn uniformly at random we can again directly embed the security challenge of the PRF challenger: we use a PRF query to compute the encryption keys as

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

If the adversary can detect whether the keys have been computed with the PRF or if the answer is a truly random value, we can directly use this adversary to break the security of the PRF.

Summarizing these changes we get that

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + 2 \cdot \epsilon_{\mathsf{prf}}.$$

Game 6. In this game we can directly plug-in the sLHAE challenge into either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. With probability $\geq 1/2$ our random choice is correct (i.e. the adversary attacks the sLHAE ciphertexts generated under this key). We have

$$\mathsf{Adv}_5 \leq 2\ell \cdot \mathsf{Adv}_6.$$

This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the $\mathsf{Encrypt}$ query of the sLHAE game. Similarly the $\mathsf{Decrypt}$ query is used to decrypt all these queries on behalf of the receiving oracle. If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then the challenger forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the challenger essentially relays all messages it is easy to see that an attacker $\mathcal{A}$ having advantage ϵ' yields an attacker 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

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

Collecting probabilities we get that

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \ell^2 \cdot \epsilon_{\mathsf{ddh}} + 2d\ell^2 \cdot (\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{slhae}}).$$

□

8 Proof Sketches

In this section we give proof sketches for the remaining ciphersuites, mutually authenticated TLS-RSA and server-only authenticated TLS-DH and TLS-DHE.

8.1 Proof Sketch for Mutually Authenticated TLS-RSA

Theorem 3. *Let μ be the output length of the PRF, ν be the length of the premaster secret, and λ be the length of the nonces r_C and r_S. Assume that the PRF is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the public key encryption scheme is $(t, \epsilon_{\mathsf{pke}}, q_{\mathsf{pke}})$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the signature scheme used for generating certificates is $(t, \epsilon_{\mathsf{ca}}, q_{\mathsf{ca}})$-secure, the signature scheme used to sign messages in TLS is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, and the sLHAE scheme is $(t, \epsilon_{\mathsf{slhae}})$-secure.*

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-RSA Handshake protocol with mutual authentication in the sense of Definition 11 with $t \approx t'$ and $q_{\mathsf{pke}}, q_{\mathsf{sig}} \geq d$, $q_{\mathsf{prf}} \geq 2$, and $q_{\mathsf{ca}} \geq w + \ell$ it holds that

$$\epsilon_{\mathsf{tls}} \leq 4\epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-2}} + 2\ell \cdot \epsilon_{\mathsf{sig}} + 16d\ell^2(2^{-\nu} + 2^{-\mu} + \epsilon_{\mathsf{H}}) + (16d\ell^2 + d\ell)(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{slhae}}).$$

When proving mutual authentication we consider the following three types of adversaries.

Type 1. The adversary breaks the authentication property of Definition 11 and the first oracle that maliciously accepts is a **client** oracle (i.e. an oracle with $\rho = \mathsf{Client}$).

Type 2. The adversary breaks the authentication property of Definition 11 and the first oracle that maliciously accepts is a **server** oracle (i.e. an oracle with $\rho = \mathsf{Server}$).

Type 3. The adversary answers the encryption challenge correctly for sessions in which client and server have mutual authentication.

The proof is similar to the proof of Theorem 1 (for TLS-RSA with server-only authentication) given in Section 6. We can argue that no adversary of **Type 1** exists (except for some negligible error probability) because we have already shown in the proof of Theorem 1 that the client only accepts if the messages sent by the client *and* server have not been modified in transit, even without authentication of the client. The only additional message sent from the client to the server oracle in the mutual authentication setting is the Certificate Verify message m_9 which contains a fresh signature (that is also computed over the random nonces). However, we can argue that any modification to the signature on transit results in the client to not accept. This is because the Server Finished message is computed over all previous messages, which would then be different for the client (which re-computes it for verification over the unmodified signature it sent) and the server (which computes it over the modified signature it received) oracle.

To prove that no adversary of **Type 2** exists (again except for some negligible error probability) we now exploit that the random nonces, contained in messages m_1 and m_2, and the encryption of the premaster secret, message m_8, are signed by the client. If the **Type 2** adversary modifies any of these messages, we can directly use the adversary to output a signature forgery. We can now substitute m_8 by the encryption of a truly random value under the assumption that the public key encryption system is CCA secure. As before, the adversary now cannot gain any information on pms from the ciphertext. Based on this modification and the security of the PRF we can in the next step substitute the master secret ms by a truly random value. With a truly random master secret we can now, again by the security of the PRF, substitute the Finished messages

$fin_C^{(S)}$ and $fin_S^{(S)}$ and the encryption keys $k_C^{(S)}$ and $k_S^{(S)}$ with truly random values. We can now embed the sLHAE key into $k_C^{(S)}$. Since the nonces and m_8 are protected from adversarial modifications, we must have that $k_C^{(S)} = k_C^{(C)}$. Now there are two cases. If the adversary modifies any of the messages m_1 to m_{10}, the Server Finished messages $fin_S^{(S)}$ and $fin_S^{(C)}$ can, exploiting the security of the hash function and the PRF, be substituted by uniformly random values which are *distinct* with probability $1-2^{-\mu}$. The message m_{11} output by the client oracle thus cannot help the adversary to make the server oracle accept. The adversary must compute an encryption of $fin_C^{(S)}$ on its own. Otherwise, in case m_{11} (Client Finished) is the only message modified by the adversary, they will be substituted by the same random value, i.e. $fin_C^{(S)} = fin_C^{(C)}$. In such a case, any adversary that makes the server oracle accept maliciously has to compute a new encryption m'_{11} of $fin_C^{(S)} = fin_C^{(C)}$. In both cases we can use the $\mathsf{Encrypt}$ query from the sLHAE security game to generate m_{11}. Therefore, to make the server accept, the adversary must now break the security of the sLHAE scheme.

To prove that no adversary of **Type 3** exists (again except for some negligible error probability) we exploit that no message up to, but not including, m_{12} has been modified on transit. Then the keys for the sLHAE scheme can be substituted by uniformly random values and we can directly plugin the sLHAE challenge in one of the encryption keys into either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the $\mathsf{Encrypt}$ query of the sLHAE game. Similarly the $\mathsf{Decrypt}$ query is used to decrypt all these queries on behalf of the receiving oracle. An adversary that solves the encryption challenge can so directly be used to break the security of the sLHAE scheme.

8.2 Proof Sketch for Server-Only Authenticated TLS-DHE

Theorem 4. *Let μ be the output length of the PRF and λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the signature scheme used for generating certificates is $(t, \epsilon_{\mathsf{ca}}, q_{\mathsf{ca}})$-secure, the signature scheme used to sign messages in TLS is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, the sLHAE scheme is $(t, \epsilon_{\mathsf{slhae}})$-secure, and the PRF-ODH-problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and* PRF.

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-DHE Handshake protocol with server-only authentication in the sense of Definition 12 (with perfect forward secrecy) with $t \approx t'$ and $q_{\mathsf{prf}} \geq 1$, $q_{\mathsf{sig}} \geq d$, and $q_{\mathsf{ca}} \geq w + \ell$ it holds that

$$\epsilon_{\mathsf{tls}} \leq 2 \cdot \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-1}} + 2\ell \cdot \epsilon_{\mathsf{sig}} + 8(d\ell)^2 \cdot \left(2^{-\mu} + 2^{-\nu} + \epsilon_{\mathsf{H}}\right) + 9(d\ell)^2 \cdot \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{slhae}}\right).$$

When proving server-only authentication we consider two types of adversaries.

Type 1. The adversary breaks the server-only authentication property of Definition 12 for a client oracle.

Type 2. The adversary answers the encryption challenge correctly for clients that do not accept maliciously.

To prove that no adversary of **Type 1** exists (again except for some negligible error probability) we proceed as follows. At first we exclude certificate forgeries and ensure uniqueness of the random nonces. Then we exclude modifications of the random nonces and the server share by reducing security to that of the signature scheme, since the server signature is computed over all these values. To do so, technically we first guess the server party j for which the adversary outputs a signature forgery with probability $\frac{1}{\ell}$. We replace the public key pk_j of this party with the challenge public key pk' of the signature security game.

For simulation of oracles of party P_j we use the signature oracle granted in the signature security game. We then guess the server oracle π_j^t and its partner (client) oracle π_i^s with probability $1/(d^2\ell)$. (Note that we have already guessed the server party holding the server oracle.)

We now consider two cases: either the client Diffie-Hellman share has been modified by the adversary (**Case 2**) or not (**Case 1**). In **Case 1** we directly substitute the master secret ms that is computed between oracles π_i^s and π_j^t by a truly random value. Any adversary that can recognize our modification can be used to break the PRF-ODH assumption.[26] To this end consider that we send the concatenation of the fixed label $label_1$ and the random nonces to the PRF-ODH challenger who responds with z_b, g^u and g^v. (For details see Definition 6.) Now we first embed the g^v share from the PRF-ODH challenge in the message m_4 sent by π_j^t. Then we set g^u to be the Diffie-Hellman share of π_i^s included in message m_8 and use z_b as the master secret for the client and server oracle. Note that g^u and g^v are distributed exactly as before, since they are chosen at random. Also we now use the PRF-ODH oracle to simulate that π_j^t has access to the secret key v (in case the adversary modifies m_8). Now if z_b is the real output of PRF this corresponds to the situation where ms is computed according to the protocol specification. However if z_b is random this exactly corresponds to the situation where ms is drawn uniformly random. So under the PRF-ODH assumption no adversary can notice our substitution of the master secret with a truly random value in **Case 1**. The remaining part of the proof is similar to that of TLS-RSA with server-only authentication: we now again consider two subcases of **Case 1**, **Case 1.1.1** and **Case 1.1.2**. Observe that in this proof we do not require two additional subcases **Case 1.1** and **Case 1.2** since the random nonces are protected by adversarial modifications via the signatures. Either one of the messages m_1 to m_{11} have been modified in transit (**Case 1.1.2**) or not (**Case 1.1.1**). In **Case 1.1.1** we thus assume that only the message m_{12} has been modified. In this case the adversary has to compute a new encryption of $fin_S^{(C)} = fin_S^{(S)}$ to make the client accept which breaks the security of the sLHAE scheme. If the adversary modifies m_1 to m_{11}, the Finished messages fin_S (i.e. $fin_S^{(C)}$ and $fin_S^{(S)}$), can, exploiting the security of the hash function and the PRF, be substituted with random values that are distinct with probability at least $1 - 2^{-\mu}$. So in **Case 1.1.2**, $fin_S^{(S)}$ is independent from $fin_S^{(C)}$ and the adversary cannot use the output of the server oracle to compute m_{12} such that the client accepts and has to compute this message on its own. However, this breaks the security of the sLHAE scheme. In **Case 1.1.1** and **Case 1.1.2** we use the $\mathsf{Encrypt}$ queries to compute the output of the server oracle m_{12}.

The **Cases 2** can be reduced to the security of the sLHAE scheme. We embed the sLHAE challenge in the key $k_S^{(C)}$ of the client oracle that is used to decrypt server messages. As in previous proofs $k_S^{(C)}$ is independent of the adversary and all computations made by the server oracle. However, now any message that makes the client accept breaks the security of the sLHAE scheme.

To prove that no adversary of **Type 2** exists (again except for some negligible error probability), we exploit that in the previous proof we technically also showed, that the client will not accept if any of the messages sent by the client will be modified. Since any message that is used to derive the encryption keys will not be modified, the encryption keys of client and server oracle are equal. With the same arguments as before, we can now substitute the encryption keys by random values and directly plugin the sLHAE challenge in either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the $\mathsf{Encrypt}$ query of the sLHAE game. Similarly the $\mathsf{Decrypt}$ query is used to decrypt all these queries on behalf of the receiving oracle. Since both DH shares are drawn randomly in each session we also have perfect forward secrecy.

Remark 6. Note that $\epsilon_{\mathsf{client}}$ (i.e. the probability that a client oracle maliciously accepts) in our proof of

[26] Due to the nature of the PRF-ODH assumption we do not replace the premaster secret pms separately.

TLS-DHE with server-only authentication differs from $\epsilon_{\mathsf{client}}$ in the proof given in JKSS for TLS-DHE with mutual authentication. This is mainly due to the fact that we extended the ACCE model of JKSS to allow for arbitrary key registration of new parties. (Recall that the adversary only wins, if a client oracle π_i^s maliciously accepts after receiving a certificate that was originally created by the challenger, i.e. π_i^s has $\Pi = j$ and $id_j \in \{id_1, \ldots, id_\ell\}$.) Thus we now have to additionally ensure that the adversary cannot break the signature scheme and create valid certificates for parties P_j such that $id_j \in \{id_1, \ldots, id_\ell\}$.

8.3 Proof Sketch for Server-Only Authenticated TLS-DH

Theorem 5. *Let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, qprf)$-secure, the signature scheme used for generating certificates is $(t, \epsilon_{\mathsf{ca}}, q_{\mathsf{ca}})$-secure, the sLHAE scheme is $(t, \epsilon_{\mathsf{slhae}})$-secure, and the Strong PRF-ODH-problem is $(t, \epsilon_{\mathsf{prfodh}}, q_{\mathsf{prfodh}})$-hard with respect to G and* PRF.

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-DH Handshake protocol with server-only authentication in the sense of Definition 12 with $t \approx t'$ and $q_{\mathsf{prf}} \geq 1$, $q_{\mathsf{prfodh}} \geq d$, and $q_{\mathsf{ca}} \geq w + \ell$ it holds that

$$\epsilon_{\mathsf{tls}} \leq 2 \cdot \epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-1}} + 8(d\ell)^2 \cdot (\epsilon_{\mathsf{H}} + 2^{-\mu}) + 9(d\ell)^2 \cdot (\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{slhae}}) \,.$$

To prove the security of TLS-DH with server-only authentication we again consider the two types of adversaries against server-only authenticated TLS as described in the previous section.

To prove that no adversary of **Type 1** exists (again except for some negligible error probability) we first proceed as in the proof of TLS-DHE with server-only authentication. We first exclude modifications of the server share which is contained as a long-term key in the server certificate. An adversary able to output a valid signature (verifiable under the key used by the simulator to generate certificates) for a new DH public key could directly be used to forge a certificate. We then guess the server oracle π_j^t and its partner (client) oracle π_i^s with probability $1/(d\ell)^2$.

Now we again follow the outline of the proof of TLS-DHE with server-only authentication and consider two cases. We have the same cases and subcases as before. Either the adversary does modify the client share (**Case 2**) or not (**Case 1**). Either the adversary does not modify any of the nonces in transit (**Case 1.1**), or it does (**Case 1.2**) so. Either the adversary does not modify any of the remaining messages m_1 to m_{11} in transit (**Case 1.1.1**), or it does (**Case 1.1.2**) so.

In **Case 1.2**, **Case 1.1.1**, and **Case 1.1.2** we exploit the Strong PRF-ODH assumption to show that the master secret is indistinguishable from random. As before, in **Case 2** the master secret $ms^{(C)}$ computed by the client oracle is indistinguishable from random and with overwhelming probability distinct from $ms^{(S)}$.

To show that the master secret is indistinguishable from random in the subcases of **Case 1**, we need to exploit the Strong PRF-ODH problem. We need the enhanced (as compared to the original PRF-ODH assumption) capabilities of the adversary in the Strong PRF-ODH game because any **Type 1** adversary can try to engage in *several (q_{prfodh})* sessions with the *same server-oracle* and the *same client share* (only using distinct nonces). Thus, the attack capabilities granted in the original PRF-ODH game do not suffice to correctly simulate the behaviour of the oracles.

In **Case 1** we substitute the master secret that is computed between oracles π_i^s and π_j^t by a truly random value. Any adversary that can recognize our modification can be used to break the Strong PRF-ODH assumption. Again consider that we send the concatenation of the first label and the random nonces to the PRF-ODH challenger, who responds with z_b, g^u and g^v. At P_j we first embed the g^v share from the PRF-ODH challenge in the certificate (without changing the distribution, since g^v is random). Also we now

use the PRF-ODH challenger to simulate that oracles of P_j have access to the secret key v. In the communication between oracles π_i^s and π_j^t, we set g^u to be the Diffie-Hellman share of π_i^s and use z_b as the master secret for the client and server oracle. Again, g^u is distributed exactly as before. We can now simulate all adversarial queries to P_j including those where the adversary re-uses g^u as the client-share. This is because the nonce generated by the server is always fresh and thus the message input to the Strong PRF-ODH oracle is distinct from the values used in the first step of the Strong PRF-ODH security game. Now, if z_b is the real output of PRF this corresponds to the situation where ms is computed honestly, if it is random, ms is random too. So under the PRF-ODH assumption no adversary can notice our substitution of the master secret with a random value in **Case 1.1.1**, **Case 1.1.2**, and **Case 1.2**. The remaining part of the proof is again similar to that of TLS-DHE with server-only authentication.

The **Cases 2, 1.2** and **1.1.2** can all be reduced to the security of the sLHAE scheme. We embed the sLHAE challenge in the key $k_S^{(C)}$ of the client oracle that is used to decrypt server messages. Since $fin_S^{(S)}$ is independent from $fin_S^{(C)}$, the adversary cannot use the output of the server oracle to compute m_{12} such that the client accepts. Thus it has to compute this message on its own. However, this breaks the sLHAE assumption. In **Case 1.1.1** the adversary has to compute a new ciphertext on the same message $fin_S^{(S)} = fin_S^{(C)}$ to make the client accept. However this also breaks the security of the sLHAE scheme.

To prove that no adversary of **Type 2** exists (again except for some negligible error probability), we proceed similar to the previous proof of TLS-DHE with server-only authentication. We first exploit that the client will not accept if any of the messages is modified. Next, we can directly plug-in the sLHAE challenge into either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the Encrypt query of the sLHAE game. Similarly the Decrypt query is used to decrypt all these queries on behalf of the receiving oracle. So any successful adversary breaks the security of the sLHAE game.

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] C. Adams, S. Farrell, T. Kause, and T. Mononen. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Proposed Standard), September 2005.

[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 and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 390–399. ACM Press, October / November 2006.

[7] 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.

[8] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, May 1994.

[9] 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.

[10] 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.

[11] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational), May 2006. Updated by RFC 5246.

[12] 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.

[13] 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.

[14] 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/`.

[15] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62. ACM Press, October 2011.

[16] 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.

[17] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218. ACM Press, May 1998.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, August 2005.

[24] 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.

[25] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is secure under the RSA assumption. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer, August 2001.

[26] 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.

[27] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, May / June 2006.

[28] Dennis Hofheinz and Eike Kiltz. Practical chosen ciphertext secure encryption from factoring. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 313–332. Springer, April 2009.

[29] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670. Springer, August 2009.

[30] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, August 2012.

[31] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model (full version). Cryptology ePrint Archive, Report 2011/219, last revised 2013. `http://eprint.iacr.org/2011/219`.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565. Springer, December 2005.

[38] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer, May / June 2006.

[39] 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.

[40] 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.

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

[42] M. Nystrom and B. Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986 (Informational), November 2000. Updated by RFC 5967.

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

[44] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, December 2011.

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

[46] Juliano Rizzo and Thai Duong. The CRIME Attack. In *ekoparty Security Conference 8° edition*, 2012.

[47] J. Schaad. Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF). RFC 4211 (Proposed Standard), September 2005.

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

[49] 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.

[50] 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.