On the Security of the TLS Protocol: A Systematic Analysis*

Hugo Krawczyk, Kenneth G. Paterson**, and Hoeteck Wee***

[1] IBM Research
[2] Royal Holloway, University of London
[3] George Washington University

Abstract. TLS is the most widely-used cryptographic protocol on the Internet. It comprises the TLS Handshake Protocol, responsible for authentication and key establishment, and the TLS Record Protocol, which takes care of subsequent use of those keys to protect bulk data. TLS has proved remarkably stubborn to analysis using the tools of modern cryptography. This is due in part to its complexity and its flexibility. In this paper, we present the most complete analysis to date of the TLS Handshake protocol and its application to data encryption (in the Record Protocol). We show how to extract a key-encapsulation mechanism (KEM) from the TLS Handshake Protocol, and how the security of the entire TLS protocol follows from security properties of this KEM when composed with a secure authenticated encryption scheme in the Record Protocol. The security notion we achieve is a variant of the ACCE notion recently introduced by Jager et al. (Crypto '12). Our approach enables us to analyse multiple different key establishment methods in a modular fashion, including the *first proof* of the most common deployment mode that is based on RSA PKCS #1v1.5 encryption, as well as Diffie-Hellman modes. Our results can be applied to settings where mutual authentication is provided and to the more common situation where only server authentication is applied.

Keywords: TLS, RSA, key exchange, modular proof, applied provable security

---

* An extended abstract of this work will appear in the proceedings of CRYPTO 2013. This is the full version.

** Supported by EPSRC Leadership Fellowship EP/H005455/1

*** Supported by NSF CAREER Award CNS-1237429

Table of Contents

On the Security of the TLS Protocol: A Systematic Analysis 1
Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee
1 Introduction 3
2 The TLS Handshake Protocol with Server-Only Authentication 8
3 Authenticated and Confidential Channel Establishment (ACCE) 9
4 From CCCA KEM Security to SACCE Security of TLS 15
5 TLS-RSA: Instantiations from OW-PCA 23
6 TLS-CCA: Instantiations from IND-CCA 26
7 TLS-DH: Instantiation from PRF-ODH 28
8 The TLS Handshake Protocol with Mutual Authentication 31
9 Conclusions and Discussion 36
A The TLS Handshake Protocol without Client Authentication 40
B Preliminary Definitions 43
C On the Necessity of the PRF-ODH Assumption 45

1 Introduction

TLS is the mostly widely used cryptographic protocol for secure communications on the Internet. The main purpose of TLS is to provide end-to-end security against an active, man-in-the-middle attacker. Originally deployed (as SSL) in web browsers for `https` connections, TLS is now used as a general purpose provider of secure communications to all kinds of applications: e-commerce transactions, virtual private networks (VPN), Android and iOS mobile apps [25, 26], as well as related procotols like DTLS [37, 44]. In short, TLS is one of the most important real-world deployments of cryptography that we have.

TLS in a nutshell. We begin with an informal high-level overview of the TLS protocol; a more detailed treatment is given in Section 2 and Appendix A. The TLS protocol is executed between a client and a server. It has two main constituents: the Handshake Protocol, which is responsible for key establishment and authentication; and the Record Protocol, which provides a secure channel for handling the delivery of data. The Handshake Protocol establishes the application keys, which are in turn used to encrypt application data in the Record Protocol. The TLS specification offers multiple options for key establishment mechanisms in the Handshake Protocol and for symmetric key encryption schemes in the Record Protocol. The most common configuration, to which we refer as TLS-RSA, relies on RSA PKCS #1v1.5 encryption in the Handshake Protocol. Other configurations include TLS-DH and TLS-DHE, which rely on Diffie-Hellman key exchange (the first uses a static server's DH key and an ephemeral client's key while in the latter both parties contribute ephemeral DH keys). All these configurations provide server authentication, with optional client authentication in settings where clients possess public keys as well.

Prior work on TLS. In view of its importance, TLS has long been the subject of intense research analysis, including, in chronological order, [48, 14, 43, 35, 32, 47, 20, 34, 5, 38, 29, 6, 39, 23, 42, 11, 2, 31, 36, 24, 16, 3, 12, 4]. The main, twin thrusts of this research have been to establish to what extent the TLS Handshake Protocol and the TLS Record Protocol are secure, for the respective tasks of key establishment and authentication and for providing a secure channel for delivery of data.

We now have a fairly complete understanding of the underlying cryptography for the Record Protocol, as studied in the works of Krawczyk [35] as well as Paterson, Ristenpart and Shrimpton [42]. These works demonstrated that, when carefully implemented to avoid timing and other attacks like those in [47, 20, 3], the stream-cipher and CBC encryption modes in the TLS Record Layer achieve the security notion of authenticated encryption; in fact, [42] puts forth and achieves a strengthening there-of, known as *stateful, length-hiding authenticated encryption* (sLHAE).

On the other hand, a complete analysis of the TLS Handshake Protocol remains elusive. A main obstacle is that the design of TLS violates the basic cryptographic principles of key indistinguishability and separation of key exchange and secure channels. This arises because the TLS Record Protocol overlaps with the TLS Handshake Protocol, and the application key is used to encrypt the last two messages of the Handshake Protocol (known as the `Finished` messages). As such, the TLS Handshake Protocol is deemed insecure by the existing security models for key exchange, initiated in the work of Bellare and Rogaway [9].

Several prior works [39, 32] circumvented this issue by analyzing *variants* of the TLS protocol (e.g. with a different message ordering, unencrypted `Finished` messages, or RSA-OAEP encryption). In particular, Morrissey, Smart and Warinschi [39] analyze the "Truncated TLS Handshake Protocol", where the `Finished` messages are not encrypted by the application key. An important feature of [39] is the modularity of the approach. This conceptually simplifies the protocol and the security proofs, and points the way forward for subsequent analysis. However, the end result applies to truncated TLS and

not to the real protocol. In addition, Morrissey et al. [39] model TLS-RSA under the assumption that RSA encryption is replaced with CCA-secure encryption which is provably false for RSA PKCS #1v1.5 encryption as used in TLS-RSA. The modularity theme from [39] is developed further in recent work by Brzuska et al. [16], who analyze the TLS protocol using a game-based framework that is designed to enable compositional results to be be proved, but their analysis of TLS-RSA assumes IND-CCA security for the RSA encryption, which, again, is known not to hold. Thus, unfortunately, given the high sensitivity of key exchange protocols in general (and TLS in particular) to small details, these results tell us little about TLS as used in practice.

In recent work, Jager et al. [31] put forth a new security notion — Authenticated and Confidential Channel Establishment (ACCE) security — which captures the desired security guarantees when the TLS Handshake and Record Protocols are used in tandem. (This circumvents the barrier pertaining to the separation of key exchange and secure channels.) In addition, they showed that the cryptographic core of the TLS-DHE protocol when both server and client authentication are applied satisfies ACCE security. Informally, this means the TLS Record Protocol when used with TLS-DHE as the Handshake Protocol constitutes a secure channel and guarantees authentication and privacy for data delivery between the server and the client. While this work constitutes a significant step forward in terms of realistic modeling and analysis of TLS, the TLS-DHE protocol is (currently) seldom used in practice, and client-side authentication via signatures is very rarely done.

Additional literature on analyzing the TLS Handshake Protocol include works on symbolic models, e.g. [43, 29, 11] and on security analysis of a TLS implementation via type-checking [12]. Works on simulation-based definitions and designs for key agreement and secure channel protocols include [46, 18, 19].

TLS-RSA. As noted earlier, the most commonly deployed mode of TLS, namely TLS-RSA, uses RSA PKCS #1v1.5 encryption [33]. In 1998, Bleichenbacher discovered a devastating man-in-the-middle attack on SSL, the predecessor of TLS. Specifically, Bleichenbacher presented a chosen-ciphertext attack on RSA PKCS #1v1.5 encryption [14], which in turn allows a man-in-the-middle adversary against SSL to recover the pre-master secret and thence the application keys. In fact, the attack only requires a ciphertext validity oracle. TLS, the successor to SSL, incorporates an *ad hoc fix* to thwart Bleichenbacher's attack: decryption failures are hidden from the adversary, including via some defences against timing attacks, thereby removing access to the ciphertext validity oracle.

For over a decade, the TLS Handshake Protocol (and in particular TLS-RSA) has largely resisted attacks; however, that in itself does not rule out the possibility of an attack being discovered in future. The folklore belief is that TLS-RSA is secure if we replace RSA PKCS #1v1.5 with RSA-OAEP or any other CCA-secure encryption scheme; unfortunately, only RSA PKCS #1v1.5 is standardised in TLS and used in practice. This begs the question:

Is TLS-RSA with RSA PKCS #1v1.5 encryption ACCE secure?

A partial answer to the above question was provided in the work of Jonsson and Kaliski Jr. [32]: they showed that RSA PKCS #1v1.5 encryption when augmented with the *unencrypted* TLS client `Finished` message is CCA-secure. However, their analysis was not extended to either the TLS Handshake Protocol or the full TLS protocol; furthermore, in TLS the client `Finished` message is actually encrypted with the application key. We stress that RSA PKCS #1v1.5 encryption when augmented with the *encrypted* TLS client `Finished` message is not even a CPA-secure key-encapsulation mechanism (KEM), for the same reason that the TLS Handshake Protocol violates key indistinguishability.

Proving security of TLS-RSA and beyond. We provide an affirmative answer to the above question, namely, we provide the first proof of security for the *unmodified* TLS-RSA protocol with RSA PKCS

#1v1.5 encryption in the commonly deployed setting of server-only authentication. More generally, we provide a systematic and modular analysis of the different modes of TLS, which include TLS-RSA, TLS-DH and TLS-DHE, in both the common setting of server-only authentication as well as with combined client and server authentication. We also validate the folklore belief that TLS Handshake with RSA replaced with any CCA-secure public-key encryption scheme (e.g., RSA-OAEP) is secure. We refer to such an instantiation as TLS-CCA. Following Jager et al. [31], we focus on the *cryptographic core* of TLS (see Appendix A for a discussion of what we omit). We concentrate on achieving ACCE security with appropriate modifications to handle server-only authentication (in which case we speak of SACCE security). We next present an overview of our framework, summarized in Figure 1.

1.1 Systematic Analysis of All TLS Handshake Modes

Our framework. We build an abstraction of the TLS Handshake Protocol via a generic representation using a *key-encapsulation mechanism (KEM)* (see Figure 2). Each of the TLS modes is then fully defined via a specific instantiation of the KEM. The goal is to find sufficient conditions on the KEM so that any instantiation satisfying these conditions immediately leads to a secure protocol in the sense of ACCE security (as discussed above). This approach has its roots in the work of Jonsson and Kaliski [32] that studied the underlying KEM in TLS-RSA.

We formalize this statement using the existing notion of constrained CCA (CCCA) security, introduced by Hofheinz and Kiltz [30] in the context of hybrid encryption. In the CCCA security game, the adversary is provided with a "constrained decryption oracle" that takes as input a pair (C, T) where C is a ciphertext and T is some auxiliary information; the oracle returns the decryption K of C if C is different from the challenge ciphertext and (K, T) satisfies some specified predicate, and $\perp$ otherwise. In particular, if the oracle returns $\perp$, the adversary does not learn whether it is because K is $\perp$ or because (K, T) fails to satisfy the predicate. In our framework, we consider CCCA security where T is an encrypted TLS client `Finished` message, and the predicate enforces validity of T. Now, if the constrained decryption oracle returns $\perp$ on query (C, T), the adversary does not learn whether it is because C is an invalid ciphertext or because T is an invalid `Finished` message – this precisely captures the intention of the TLS fix for thwarting Bleichenbacher's attack! We note that the challenge ciphertext in the CCCA security experiment is not accompanied by the corresponding `Finished` message; this asymmetry between the challenge ciphertext and the oracle queries allows us to bypass the key indistinguishability barrier in TLS.

ACCE Security from CCCA Security. Our first result says that if the key encapsulation mechanism in the TLS Handshake Protocol satisfies CCCA security and the encryption scheme used in the TLS Record Protocol is sLHAE-secure, then TLS is ACCE secure, in the server-only authentication setting. We stress that this result is in the standard model. Importantly, the CCCA security game is conceptually and technically much simpler to analyze than the whole TLS protocol, as we do not have to worry about multiple sessions, nonces, or the multiple message flows in the full protocol.

To establish ACCE security, we need to achieve security against a (concurrent) man-in-the-middle adversary communicating with multiple honest clients and multiple honest servers. Roughly speaking, we will rely on the constrained decryption oracle in CCCA security to simulate the honest servers. The main technical difficulty in establishing this result arises when a man-in-the-middle adversary plays a relaying strategy between an honest server and client and then mauls the client's encrypted `Finished` message. Here, we cannot rely on the constrained decryption oracle to simulate the honest server's response because the adversary is using the challenge ciphertext. Moreover, we cannot immediately appeal to the non-malleability of the sLHAE-secure scheme used to encrypt the `Finished` message since the protocol messages leak information about the application key. To solve this problem, we exploit

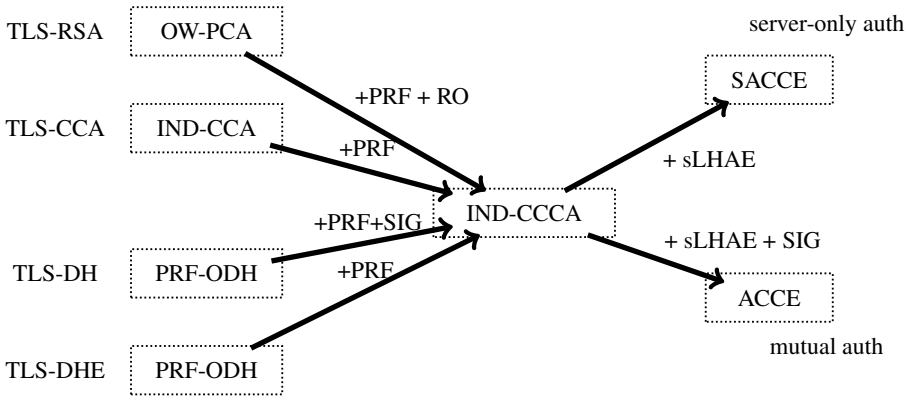

Fig. 1. Summary of our results.

the fact that the CCCA security game provides us with a real-or-random key K^*, which we may use to decrypt and verify the client's encrypted `Finished` message for this specific adversarial strategy. We stress that this techical difficulty goes away if the client's `Finished` message is unencrypted, because the prior transcript uniquely determines an accepting `Finished` message.

CCCA Security in the TLS Handshake. Our second set of results says that the key encapsulation mechanisms underlying the TLS-RSA, TLS-CCA, TLS-DH, and TLS-DHE variants of the TLS Handshake Protocol all satisfy CCCA security. Combined with our first result, this yields ACCE security of TLS-RSA, TLS-CCA, TLS-DH and TLS-DHE (see Figure 1). We proceed to outline each of these results:

- **TLS-RSA.** The proof of CCCA security of TLS-RSA KEM uses the same assumptions and similar techniques to those in [32]. We require that RSA PKCS #1v1.5 encryption is OW-PCA-secure (one-way against plaintext checking attacks), and the result is in the random oracle model. In turn, the latter can be proved under a variant of the RSA assumption introduced in [32] and discussed below.
- **TLS-CCA.** This refers to TLS in the setting where the KEM is CCA-secure, e.g. when RSA PKCS #1v1.5 in TLS-RSA is replaced with RSA-OAEP. Here, we can obtain a result in the standard model. In the proof, we have to address a technical subtlety similar to the one mentioned above for the reduction from CCCA to ACCE security. Here, we exploit the security of a pseudorandom function where an encryption of the seed is given to the adversary.
- **TLS-DH and TLS-DHE.** For TLS-DH and TLS-DHE, the underlying KEMs are variants of the ElGamal encryption scheme. In the TLS-DH case, we establish the KEM's CCCA security (in the standard model) under a variant of the PRF-ODH assumption that was introduced in [31] for analyzing TLS-DHE. For TLS-DHE, the original PRF-ODH assumption from [31] suffices for our analysis. As an alternative, we also prove the CCCA of these KEMs under the Strong DH assumption in the random oracle model.

ACCE Security of TLS with Mutual Authentication. We extend the above results, developed for the case of server-only authentication, to the case of mutual authentication, namely, when the client authenticates itself via a digital signature. We show that also in this setting, CCCA security of the underlying KEM implies ACCE security with both server and client authentication. The extension is relatively straightforward (a positive feature!) requiring minor changes to the server-authentication-only proofs of server authentication and channel security, and the addition of a client authentication proof. The resultant analysis is generic and independent of the different underlying KEM instantiations, thus

it directly applies to TLS-RSA, TLS-CCA, TLS-DH and TLS-DHE (demonstrating the power of our modular analysis).

1.2 Summary of Results

As a result of the above methodology we obtain proofs of ACCE security for the TLS handshake protocol for *all* of the above TLS modes, both in the common setting of server-only authentication as well as with mutual authentication. These results are depicted in Figure 1 and are enumerated here with the assumptions used in each case. In all cases we assume a secure PRF and the TLS Record Protocol encryption implemented with an sLHAE encryption scheme. For the case of ACCE security with mutual authentication a secure client signature is also assumed. Certificates for both servers and clients are assumed to be provided by a minimally trusted CA that faithfully checks identities before issuing certificates. No other checks from the CA (such as proofs of possession, uniqueness of public keys, etc.) are assumed.

TLS-RSA. We obtain the first proof of security of TLS-RSA as deployed in practice, with RSA PKCS #1v1.5 and server-only authentication, in the random oracle model and under the assumption that RSA PKCS #1v1.5 is OW-PCA secure. The latter assumption, formalized in Section 5, states that inverting the encryption function is hard even given an oracle that on input a plaintext-ciphertext pair (K, ψ) checks whether the decryption of ψ equals K (for $K \neq \perp$). The OW-PCA security of RSA PKCS #1v1.5 can be proven under an RSA-like assumption, known as "partial-domain RSA with decision oracle", introduced by Jonsson and Kaliski in [32] and which we present in Section 5.2. We refer to [32] for a discussion on why this assumption is reasonable for typical parameters used in TLS; to the best of our knowledge no weakness in this assumption has been discovered since its introduction in [32]. When clients authenticate in TLS-RSA using digital signatures then full ACCE (i.e. with mutual authentication) is proven assuming a secure signature scheme. We stress that TLS-RSA is the only TLS mode whose proof is in the random oracle model; we prove all other modes in the standard model.

TLS-CCA. We prove that when instantiated with a CCA-secure public-key encryption scheme (instead of RSA PKCS #1v1.5), TLS is ACCE secure in the standard model. While no such schemes are currently standardised for TLS, this result confirms the intuition that IND-CCA security is the "right" target for the public key encryption scheme used in TLS. It also means that, should the current RSA-based encryption scheme used in TLS ever be replaced by a CCA-secure one, then our analysis will immediately provide strong security guarantees for the protocol.

TLS-DH and TLS-DHE. We prove ACCE security (with and without client authentication) of TLS-DH in the standard model under the PRF-ODH assumption introduced in [31].[4] The PRF-ODH assumption rules out some potential related-key attacks on the Kdf function that would render the protocol insecure. In Appendix C we show this assumption to be *provably necessary* for the security of TLS-DH, showing attacks on the protocol with PRFs for which the assumption does not hold. We note that we can also prove TLS-DH in the random oracle model under the Strong DH assumption. Finally, we obtain security for TLS-DHE as a corollary of our results for TLS-DH security, under the PRF-ODH assumption as well as secure signatures for servers (and clients in the case of mutual authentication). Note that our results for TLS-DHE do not encompass forward security, but this is guaranteed by the results of [31].

Discussion. In Section 9 we provide an extensive discussion of the implications of our work for TLS, and what it may teach us about secure protocol design more generally.

[4] The assumption is a variant of the ODH assumption from [1] where the oracle is implemented via a PRF rather than by a hash function. In the proof of TLS-DH we require security against multiple oracle queries while for TLS-DHE a single query suffices, as was the case in [31].

2 The TLS Handshake Protocol with Server-Only Authentication

In this section, we present our model of the TLS Handshake Protocol when no client authentication takes place. As noted in the introduction, this includes TLS-RSA, the most common usage of the TLS protocol. The parties to the protocol are a client C and a server S. Each maintains an internal state variable ST and $\Lambda \in \{\emptyset, \mathtt{acc}, \mathtt{rej}\}$. The protocol makes use of a number of cryptographic components: a key derivation function (KDF) Kdf, a pseudorandom function PRF, a stateful authenticated encryption with associated data (AEAD) scheme $\mathsf{stE} = (\mathsf{stE.Gen}, \mathsf{stE.Init}, \mathsf{stE.Enc}, \mathsf{stE.Dec})$, and a KEM $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$. The protocol is shown schematically in Figure 2. We also describe below how the keys established by this protocol are subsequently used by the TLS Record Protocol.

The model is derived from the current TLS specification [22], and we believe that our model captures the *cryptographic core* of TLS. It has a comparable level of accuracy to the model of TLS-DHE used in [31]. We highlight several salient properties of our model, and defer a detailed justification and discussion to Appendix A:

- We assume that the ciphersuites, KDF, PRF, and the stateful AEAD scheme are fixed once and for all. We do not model ciphersuite negotiation/renegotiation, nor session resumption. In particular, this means that, while our treatment covers multiple ciphersuites (such as those based on RSA key transport and various Diffie-Hellman (DH) ciphersuites) in a modular fashion, our analysis currently does not treat the case where different protocols runs may negotiate different ciphersuites. This requires the application of a suitable composability framework that is beyond the immediate scope of this paper.
- In the case of TLS-RSA, $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ represents the algorithms of the RSA PKCS#1v1.5 encryption scheme (c.f. Section 5.2). The specifics of this encoding were analysed in detail in [14, 32]. For this mode, we assume that the outcome of processing CRES at the server end is completely hidden from the adversary. Such an assumption is necessary; otherwise, TLS-RSA is susceptible to Bleichenbacher's attack [14]. Formally, we model this by treating $\text{CRES}\|\text{CFIN}$ as a monolithic message in the proof of security.
- Our generic description includes the TLS-DH mode, where the server has a certificate on a static DH key PK_S and DH key exchange is used to establish PMS. Here $(\text{CRES}, \text{PMS}) \leftarrow \mathsf{F}_C(\text{PK}_S)$ denotes the client's computation of an ephemeral DH value (CRES) and the pre-master secret (PMS); $\text{PMS} \leftarrow \mathsf{F}_S(\text{SK}_S, \text{CRES})$ denotes the corresponding computation on the server side. In this situation, we may alternatively think of $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ as being the algorithms of a Diffie-Hellman (or Elgamal-type) KEM based on public key PK_S. See Section 7. Specifically, with appropriate choices of Diffie-Hellman groups, our analysis covers the DH_DSS, DH_RSA, ECDH_ECDSA, and ECDH_RSA key exchange methods from [22, 13]; here the suffix DSS/RSA/ECDSA has no meaning since the server does not sign in this mode.
- By suitably extending CERT_S to include the server's signature on its choice of ephemeral DH value, our description captures the TLS-DHE mode, where now PK_S is a signature verification key and PMS is the result of a DH key exchange based on the ephemeral values chosen by client and server. This then covers the DHE_DSS, DHE_RSA, ECDHE_ECDSA, and ECDHE_RSA key exchange methods from [22, 13], where the suffix DSS/RSA/ECDSA refers to the signature scheme used by the server.
- Our description also captures TLS-CCA, where $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ represents the algorithms of an IND-CCA-secure encryption scheme, such as RSA-OAEP. We note that no such encryption scheme is currently standardised for use in TLS. However, a proof of security for TLS in this case emerges cleanly from our framework and it may serve to encourage the use of CCA-secure encryption algorithms in future TLS deployments.

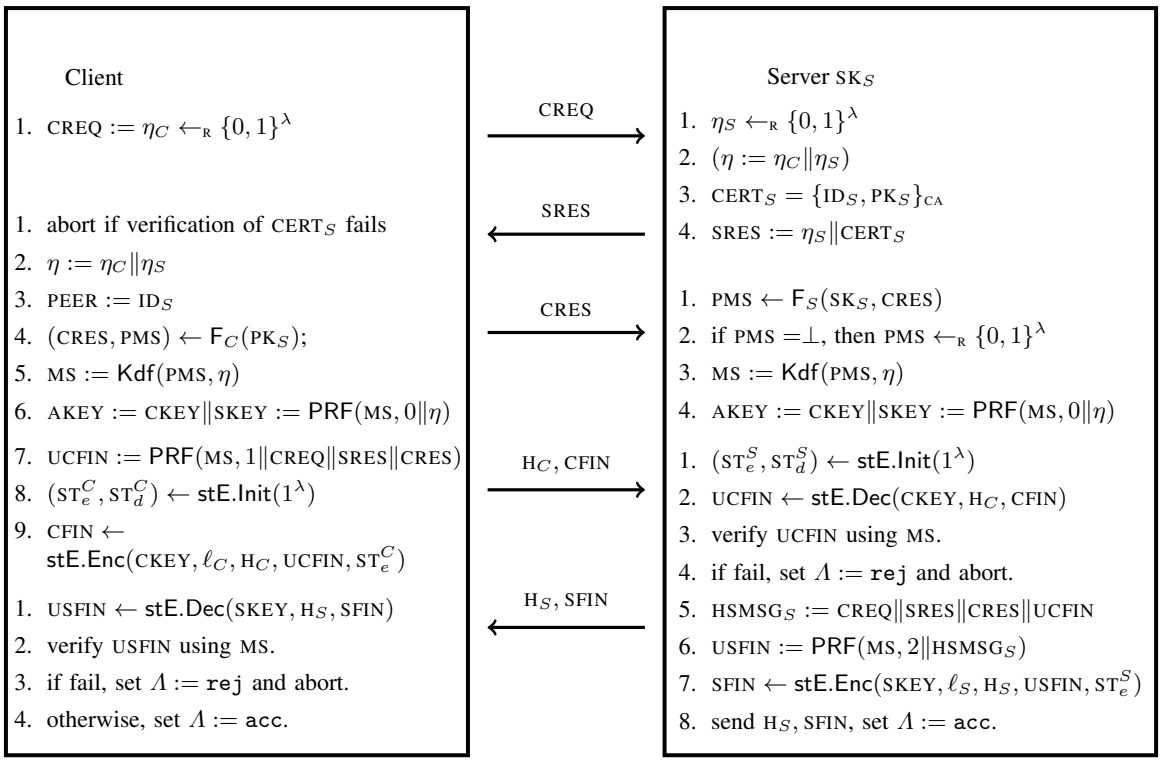

Fig. 2. Basic Generic TLS Handshake Protocol Parameterized by $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$

– For notational simplicity, we use the same symbol λ to denote the security parameter, as well as the bit length of PMS, MS, CKEY and SKEY.

TLS Record Protocol. A party that concludes the TLS Handshake protocol successfully continues to use the application key AKEY = CKEY$\|$SKEY in the TLS Record Protocol. Specifically, the client uses CKEY to encrypt messages to the server, and the server uses the same key CKEY to decrypt these messages. Similarly, the server uses SKEY to encrypt messages to the client, and the client uses the same key SKEY to decrypt these messages. As noted above, the client and server `Finished` messages are already encrypted in this way. As in [31], we model the TLS Record Protocol via a stateful AEAD scheme stE which we will assume to be sLHAE-secure. For further details, see Appendix A where we provide more specific details on the TLS specification (especially for TLS-RSA) and Appendix B.1 where we describe sLHAE-security.

3 Authenticated and Confidential Channel Establishment (ACCE)

We begin with the definition of *authenticated and confidential channel establishment (ACCE)* from [31, 9]. We will describe the syntax for a general ACCE protocol but for security, we consider a specialization to the setting where only the server is authenticated – we call this *server-only authenticated and confidential channel establishment (SACCE)*.

ACCE protocols. An ACCE protocol is a protocol executed between two parties, a client and a server. In the original description [31], an ACCE protocol has two distinct phases, called the 'pre-accept' phase and the 'post-accept' phase, corresponding to whether a party has accepted a session key in a particular

session or not. We dispense with this distinction (though it is still expressed in our security model by making the queries that are available to the adversary depend on an oracle's acceptance state). The parties in the protocol first compute as the session key an application key AKEY. Then, encrypted and authenticated data is transmitted using a symmetric encryption scheme with the application key AKEY. More specifically, AKEY is parsed as CKEY||SKEY, the client uses CKEY in a stateful AEAD scheme stE to send data to the server, and the server uses SKEY in stE to protect data sent to the client. Henceforth, we will only refer to application keys and not to session keys. Parties also maintain internal state Λ, and clients keep an additional PEER variable. We assume that an ACCE protocol is such that, when a party reaches the state $\Lambda = \mathtt{acc}$, it has already computed an application key AKEY and executed $\mathsf{stE.Init}$. TLS meets this requirement.

3.1 Execution environment

Protocol entities. Following [31, 9], we consider a set of parties $\mathcal{P} = \mathcal{S} \cup \mathcal{C}$, where $\mathcal{S}$ and $\mathcal{C}$ are disjoint and each party $P \in \mathcal{P}$ is a (potential) protocol participant. Here, $\mathcal{S}$ and $\mathcal{C}$ denote the sets of *honest* servers and clients, respectively. In addition, each $P \in \mathcal{S}$ (the servers) has a *unique*[5] key pair $(\mathrm{PK}_P, \mathrm{SK}_P)$, an identity $\mathrm{ID}_P \in \{0,1\}^\lambda$ along with a certificate $\mathrm{CERT}_P := (\mathrm{ID}_P, \mathrm{PK}_P)_{\mathrm{CA}}$ signed by a certification authority CA. We also assume that all the parties in $\mathcal{S}$ have distinct identities. (See more below about assumptions on CA and the treatment of honest vs. corrupted parties.)

Session oracles. To model several sequential and parallel executions of the protocols and sessions, each party P maintains a collection of oracles $\{\pi_1^P, \pi_2^P, \ldots\}$. The oracle π_i^P models party P executing a single instance of a protocol in "session" i. We stress that the session numbers i are just an artefact of our security model – they are designed to provide a means for the adversary to deliver messages to different sessions at different parties. In particular, the protocols and oracles need not even be "aware" of what their session numbers are (i.e. those numbers need not form part of the state).

Each oracle π_i^P maintains as internal state a set of variables comprising:

- $\Lambda \in \{\emptyset, \mathtt{acc}, \mathtt{rej}\}$;
- $\mathrm{AKEY} = \mathrm{CKEY} \| \mathrm{SKEY} \in \{0,1\}^{2\lambda}$, where $\{0,1\}^{2\lambda}$ is the application key space of the protocol;[6]
- if $P \in \mathcal{C}$, then it has an additional PEER variable to denote the intended partner (only client oracles have the PEER variable because only servers have identities);
- if $P \in \mathcal{S}$, then π_i^P also knows the party identity ID_P.

The internal state of each oracle is initialized to $(\Lambda, \mathrm{AKEY}, \mathrm{PEER}) = (\emptyset, \emptyset, \emptyset)$, where $\emptyset$ denotes undefined.

Adversarial queries. The adversary interacts with the oracles via the following queries:

$\mathsf{Send}(\pi_i^P, m)$: the adversary uses this query to send a message m to oracle π_i^P; the oracle will respond with an outgoing message according to the protocol specification and its internal state. If π_i^P has reached state $\Lambda = \mathtt{acc}$, then it replies with $\perp$. When the attacker asks the first Send-query to an oracle π_i^C where $C \in \mathcal{C}$, the oracle checks whether m is a special "Initiate client session" symbol $\top$, and if so, responds with the first protocol message (which will be a fresh client nonce). The variables Λ, AKEY are also set according to the protocol specification.

[5] Public key uniqueness is not fundamental but it simplifies presentation in some cases, e.g., when considering reduction to selective security.

[6] Here and throughout, we refer to *application* keys rather than the more usual *session* keys.

$\mathsf{Encrypt}(\pi_i^P, \ell, \mathrm{H}, m_0, m_1)$:
$u \leftarrow u + 1$
$(c^0, \mathrm{ST}_e^0) \leftarrow_{\mathrm{R}} \mathsf{stE.Enc}(K, \ell, \mathrm{H}, m_0, \mathrm{ST}_e)$
$(c^1, \mathrm{ST}_e^1) \leftarrow_{\mathrm{R}} \mathsf{stE.Enc}(K, \ell, \mathrm{H}, m_1, \mathrm{ST}_e)$
If $c_i^0 = \perp$ or $c_i^1 = \perp$ then return $\perp$
Set $c_u = c^{b_i^P}$, $\mathrm{H}_u = \mathrm{H}$ and $\mathrm{ST}_e = \mathrm{ST}_e^{b_i^P}$
Ret c_u

$\mathsf{Decrypt}(\pi_i^P, \mathrm{H}, c)$:
If $b_i^P = 0$ then Ret $\perp$
$v \leftarrow v + 1$
$(m, \mathrm{ST}_d) \leftarrow \mathsf{stE.Dec}(K, \mathrm{H}, c, \mathrm{ST}_d)$
If $v > u$ or $c \neq c_v$ or $\mathrm{H} \neq \mathrm{H}_v$ then $\mathsf{phase} \leftarrow 1$
If $\mathsf{phase} = 1$ then Ret m
Ret $\perp$

Fig. 3. The Encrypt and Decrypt oracles in the ACCE security game.

$\mathsf{Reveal}(\pi_i^P)$: the oracle π_i^P responds with the contents of the application key AKEY. Note that this query can be issued to π_i^P before it has reached state $\Lambda = \mathtt{acc}$.

$\mathsf{Encrypt}(\pi_i^P, \ell, \mathrm{H}, m_0, m_1)$: if π_i^P has *not* reached state $\Lambda = \mathtt{acc}$, this oracle returns $\perp$. Otherwise, prior to reaching state acc, π_i^P has (by assumption) computed AKEY and run the stE.Init algorithm of a stateful AEAD scheme $\mathsf{stE} = (\mathsf{stE.Gen}, \mathsf{stE.Init}, \mathsf{stE.Enc}, \mathsf{stE.Dec})$ to define states ST_e, ST_d specific to the oracle π_i^P; the game also samples a random bit b_i^P at this point, parses AKEY as $\mathrm{CKEY} \| \mathrm{SKEY}$, and adds the 5-tuple $(\mathrm{ST}_e, \mathrm{ST}_d, b_i^P, \mathrm{CKEY}, \mathrm{SKEY})$ to the oracle state ST. Now, when receiving the Encrypt query, the message $m_{b_i^P}$ is encrypted along with header data H using algorithm stE.Enc and key $K = \mathrm{CKEY}$ (if $P \in \mathcal{C}$) or key $K = \mathrm{SKEY}$ (if $P \in \mathcal{S}$) to form a ciphertext of length ℓ, and to update the encryption state ST_e. The resulting ciphertext is returned to the adversary. For details, see Figure 3.

$\mathsf{Decrypt}(\pi_i^P, \mathrm{H}, c)$: this query is intended to allow the adversary to decrypt ciphertexts that would be processed by the communication partner of the oracle π_i^P (a server S if $P \in \mathcal{C}$, and a client C if $P \in \mathcal{S}$). If π_i^P has *not* reached state $\Lambda = \mathtt{acc}$, the oracle returns $\perp$. Otherwise, when $b_i^P = 0$, the response is always $\perp$; when $b_i^P = 1$, this query involves the decryption of H and c using algorithm stE.Dec and the appropriate key K obtained from ST at the oracle: if $P \in \mathcal{C}$, this will be CKEY, and if $P \in \mathcal{S}$, then it will be SKEY. The resulting message (or failure symbol $\perp$) is returned if the query is "out-of-sync". For details, see Figure 3. Notice the similarity with Figure 6 defining sLHAE security for symmetric encryption. Essentially, the combination of Encrypt and Decrypt oracles in this ACCE game gives the adversary the same capabilities for multiple pairs of interacting parties that an adversary has in the sLHAE game for a single key. In particular, the "out-of-sync" condition (i.e. when $\mathsf{phase} \leftarrow 1$) is set in the same way as in Figure 6, in order to capture both header integrity and ciphertext integrity.

Remark 1 (on Encrypt *and* Decrypt *queries).* Note that in the TLS specification, H is constrained to contain certain values by the Record Protocol (see Appendix A for details). We do not require our adversary to respect these constraints; in this sense we give the adversary more freedom to interact with the protocol than he has in reality. Note also that our formulation only returns a single error message $\perp$ to the adversary. While the model could be extended to include multiple error messages (see for example [15]), if TLS's symmetric portion is implemented such that it has multiple, distinguishable error messages, then it is likely to be insecure. This is demonstrated by attacks in [20, 2, 3]. So assuming a single error message seems appropriate when modelling TLS, even if the attacks show that it is difficult to achieve in practice. Finally, we note that in the real TLS protocol, the functionality represented by the Decrypt oracle would not always be immediately available to the adversary, since the real entity performing the decryption may not yet have completed the Handshake and established suitable application keys. Thus our model grants the adversary slightly more power than it would have in reality.

Honest vs corrupted parties. An honest party P is one that follows the prescribed protocol and whose secrets are not accessible to the adversary except via Reveal and Test queries. We follow the convention that *an oracle π_i^P always corresponds to an honest party P*; similarly, the set $\mathcal{P} = \mathcal{S} \cup \mathcal{C}$ refers to honest parties. A corrupted party is one whose actions are controlled by the adversary. The private and public keys of corrupted parties are chosen by the adversary; for instance, it may register the key of an honest party under a corrupted party name. On the other hand, we do not allow the adversary to register a key under the identity of an honest party. In addition, our model considers *non-adaptive corruptions*, that is, the set of honest parties is determined at the onset of the security game and these remain uncorrupted for the whole game. The attacker can, however, create as many corrupted parties as it desires with any public keys of its choice. In the context of TLS, adaptive corruptions are only relevant when modeling forward secrecy and therefore do not apply to the modes analyzed here (with the exception of TLS-DHE with its ephemeral exponents). Thus, for simplicity and clarity of presentation we omit the forward secrecy extensions to the model and refer the reader to [31] for such a treatment in the TLS-DHE case.

Certificate authority. We assume that there is a single certificate authority (CA), which uses a secure signature scheme sig and whose public key is distributed to all the clients. For each $S \in \mathcal{S}$ with public key PK_S, the CA signs the pair $(\text{ID}_S, \text{PK}_S)$ to provide a certificate $\text{CERT}_S := (\text{ID}_S, \text{PK}_S)_{\text{CA}}$. We also allow the adversary access to the CA to register any number of parties, not in the set $\mathcal{S}$, with any public keys of the adversary's choice. This modeling of the CA captures the requirement that CA checks for parties' identities before issuing a certificate but checks nothing about the value or form of the public key itself (i.e., no structural or uniqueness tests, or proofs of possessions, are assumed). Assuming a unique CA is for simplicity only – having multiple CAs with their correct public keys distributed to all honest parties works as well.

Matching conversations. We consider a definition of matching conversation which is specific to TLS (and differs from the one in [31]):

Definition 1 (Matching conversations). *We say that π_i^P has a* matching conversation *with $\pi_j^{P'}$ if*

- *either $P \in \mathcal{C}$ and $P' \in \mathcal{S}$, or $P \in \mathcal{S}$ and $P' \in \mathcal{C}$; and*
- *π_i^P accepts; and*
- *the transcripts at both π_i^P and $\pi_j^{P'}$ begin with the same three messages* $(\text{CREQ}, \text{SRES}, \text{CRES})$.

Remark 2. Defining matching conversations as above means that we may treat $(\text{CREQ}, \text{SRES}, \text{CRES})$ as a post-specified session identifer. Observe that these three messages uniquely determine the parties' nonces and server's identity as well as the key PMS which in turn determines the application key. In addition, these three messages determine the client's `Finished` message, as well as the server's `Finished` message if the server reaches the accept state.

3.2 Correctness and Security

Correctness. For every honest $C \in \mathcal{C}$ and $S \in \mathcal{S}$, if two sessions π_i^C, π_j^S have matching conversations with each other (and thus both oracles reach the accept state), then we require that they output same application key AKEY and π_i^C has its PEER variable set to ID_S. We also require that the encryption scheme stE used to model the secure channel is correct.

SACCE Security. Security of an ACCE protocol with server-only authentication (SACCE) is defined by requiring that (i) the protocol provides server authentication (but with no guarantee of client authentication[7]), and that (ii) the subsequent use of the application keys in the stateful AEAD scheme stE provides stateful Length Hiding Authenticated Encryption (sLHAE), as per [42] and Appendix B.1. Informally, consider an adversary that tries to break the privacy and/or authenticity of client data in the TLS Record Protocol. Roughly speaking, the first condition says that the only way to get a client to start transmitting data in the Record Protocol is to play a relaying strategy with some honest server (leading to a matching conversation). The second says that if the adversary plays a relaying strategy between an honest server-client pair, then that pair of parties would have established a secure channel in the Record Protocol.

The formal definition is captured in terms of a game played between the adversary $\mathcal{A}$ and a challenger. This game is obtained by adapting [31, Definition 7] to our setting. At the beginning of the game, the challenger generates the long-term key-pair $(\mathrm{PK}_S, \mathrm{SK}_S)$ along with the certificate $\mathrm{CERT}_S := (\mathrm{ID}_S, \mathrm{PK}_S)_{\mathrm{CA}}$ for all $S \in \mathcal{S}$ and gives all the certificates to $\mathcal{A}$ as input. Now the adversary issues a sequence of queries defined before. The challenger answers all queries to π_i^C by running the honest client protocol, and all queries to π_j^S by running the honest server protocol using the key SK_S. The challenger will also provide certificates along with signatures to the adversary for any identities outside the set $\{\mathrm{ID}_S : S \in \mathcal{S}\}$.

Advantage measures. We associate to an adversary $\mathcal{A}$ against an ACCE protocol Π two advantage measures:

- (server authentication, i.e. client accepts $\Rightarrow$ matching conversations.)
  $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-sa}}(\mathcal{A})$ is the probability that when $\mathcal{A}$ terminates, there is a (honest) client C and oracle π_i^C that reaches an accept state with honest $\mathrm{PEER} = \mathrm{ID}_S$, but there is no unique oracle π_j^S for which π_i^C has had a matching conversation with π_j^S.
- (channel security.)
  $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-ae}}(\mathcal{A})$ is defined to be $p - 1/2$, where p is the probability that $\mathcal{A}$ outputs (P, i, b') such that $b' = b_i^P$ where b_i^P is set during the $\mathsf{Encrypt}(\pi_i^P, \ldots)$ query and we define b' to be $\perp$ unless the following conditions hold:
  1. π_i^P reaches an accept state;
  2. π_i^P is not the subject of a Reveal query; and if there is an oracle $\pi_j^{P'}$ with which π_i^P has a matching conversation then $\pi_j^{P'}$ is not the subject of a Reveal query either.
  3. $P \in \mathcal{C}$.

  The second condition serves to ensure that $\mathcal{A}$ will not win via the "trivial attack" in which it learns the application key via a Reveal query against π_i^P or a matching $\pi_j^{P'}$. The third condition is specific to the SACCE setting and it means that we only consider $\mathsf{Decrypt}$ queries issued against an (accepting) session at an honest client. Because of the lack of client authentication in this model, there is no guarantee of security for a session π_j^S at a server, unless there is an honest client session π_i^C that has a matching conversation with π_j^S. In the latter case, one can assume (without loss of generality) that the $\mathsf{Decrypt}$ query is issued against π_i^C.

Definition 2 (SACCE-secure). *We say that an ACCE protocol Π is* SACCE-secure *if Π satisfies correctness, and for all PPT adversaries $\mathcal{A}$, both $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-sa}}(\mathcal{A})$ and $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-ae}}(\mathcal{A})$ are a negligible function of the security parameter λ.*

[7] We extend the model with client authentication in Section 8.

3.3 Selective Security

We introduce the notion of *selective SACCE security* that helps to simplify our proofs.

Selective server authentication. We require that at the beginning of the security game, the adversary "commits" to (C^*, S^*, i^*) and they must correspond to (C, S, i) in the definition of $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-sa}}(\mathcal{A})$.

Selective channel security. We require that at the beginning of the security game for $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-ae}}(\mathcal{A})$, the adversary "commits" to (C^*, S^*, i^*, j^*) where (C^*, i^*) correspond (P, i) in the definition of $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-ae}}(\mathcal{A})$ and S^*, j^* are such that $\pi_{i^*}^{C^*}$ has a matching conversation with $\pi_{j^*}^{S^*}$ (the existence of such matching $\pi_{j^*}^{S^*}$ will follow from the proof of server authentication that will precede the proof of channel security).

We use $\mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-sa}}(\mathcal{A}), \mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-ae}}(\mathcal{A})$ to denote the respective advantage measures in the selective setting.

Lemma 1. *For any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}_{\Pi}^{\mathrm{sacce-sa}}(\mathcal{A}) \leq d_{max} \cdot |\mathcal{S}| \cdot |\mathcal{C}| \cdot \mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-sa}}(\mathcal{B})$$
$$\mathbf{Adv}_{\Pi}^{\mathrm{sacce-ae}}(\mathcal{A}) \leq d_{max} \cdot |\mathcal{S}| \cdot |\mathcal{C}| \cdot (d_{max} \cdot \mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-ae}}(\mathcal{B}) + \mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-sa}}(\mathcal{B}))$$

where d_{max} is an upper bound on the number of sessions invoked by $\mathcal{A}$ at any party. Moreover, the running time of $\mathcal{B}$ is roughly that of $\mathcal{A}$.

In particular, if both $\mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-sa}}(\mathcal{B})$ and $\mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-ae}}(\mathcal{B})$ are negligible, then both $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-sa}}(\mathcal{A})$ and $\mathbf{Adv}_{\Pi}^{\mathrm{sacce-ae}}(\mathcal{A})$ are negligible.

Proof (sketch). The proof follows via a standard reduction, where $\mathcal{B}$ simulates $\mathcal{A}$ as follows:

- $\mathcal{B}$ picks $C^* \leftarrow_{\mathrm{R}} \mathcal{C}$ and $S^* \leftarrow_{\mathrm{R}} \mathcal{S}$ in advance; with probability $(|\mathcal{C}| \cdot |\mathcal{S}|)^{-1}$, they match the (C, S) chosen by $\mathcal{A}$.
- Next, $\mathcal{B}$ matches the public key of S^* to that of the single server, and generates key pairs for all remaining servers so that it may simulate all of these servers.
- $\mathcal{B}$ proceeds to pick $i^* \leftarrow_{\mathrm{R}} [d_{\mathrm{max}}]$
- For channel security, $\mathcal{B}$ also picks $j^* \leftarrow_{\mathrm{R}} [d_{\mathrm{max}}]$, with the extra $\mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-ae}}(\mathcal{B})$ term to account for the probability that C_i^* accepts without any matching conversation to some π_j^S.

The claim follows readily. □

It follows readily from the Lemma that an ACCE protocol Π is *SACCE-secure* if Π satisfies correctness and for all PPT adversaries $\mathcal{A}$, both $\mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-sa}}(\mathcal{A})$ and $\mathbf{Adv}_{\Pi}^{\mathrm{s-sacce-ae}}(\mathcal{A})$ are a negligible function of the security parameter λ.

Important simplifications and conventions. In our proofs we will assume the existence of *a single honest client and a single honest server*, which correspond to C^* and S^* in the selective experiment. This simplification is without loss of generality since we may simply simulate all clients and servers different from C^* and S^*. To further simplify notation we will refer to these honest parties as C and S (rather than C^*, S^*). Thus, the only oracles we need to consider are of the form π_i^C and π_j^S for $i, j \in [d_{\mathrm{max}}]$.

4 From CCCA KEM Security to SACCE Security of TLS

In this section we state and prove the following theorem which is our core intermediate result for proving ACCE security of all TLS modes. It uses the notion of CCCA security and the definition of the TLS KEM tlskem introduced below in Sections 4.1 and 4.2, respectively.

Theorem 1. *If* tlskem *is IND-CCCA secure,* casig *is an existentially unforgeable signature scheme and* stE *is sLHAE-secure then TLS is SACCE-secure.*

The IND-CCCA security of the KEMs arising from all TLS modes (and hence the SACCE security of these modes) is shown in the subsequent sections.

4.1 IND-CCCA Security

We consider a variant of IND-CCCA security from [30]:

Definition 3 (IND-CCCA). *For a stateful adversary $\mathcal{A}$, an LKEM* lkem *and a predicate* pred*, we define the advantage function*

$$\mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{lkem},\mathsf{pred}}(\mathcal{A}) := \Pr\left[b = b' : \begin{array}{l} (\mathrm{PK},\mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda); \\ L^* \leftarrow \mathcal{A}^{\mathsf{CDec}(\mathrm{SK},\cdot,\cdot,\cdot)}(\mathrm{PK}); \\ (C^*,K^*) \leftarrow \mathsf{Enc}(\mathrm{PK},L^*); \\ K_0 := K^*; K_1 \leftarrow_{\mathrm{R}} \{0,1\}^\lambda; b \leftarrow_{\mathrm{R}} \{0,1\}; \\ b' \leftarrow \mathcal{A}^{\mathsf{CDec}(\mathrm{SK},\cdot,\cdot,\cdot)}(C^*,K_b) \end{array} \right] - \frac{1}{2}$$

with the restriction that (1) L^* must be different from all previously queried L, and (2) the restriction on the decryption oracle for queries after getting the challenge ciphertext is $(L,C) \neq (L^*,C^*)$; and where the "constrained" decryption oracle CDec *is given by:*

$$\begin{array}{l} \mathsf{CDec}(\mathrm{SK},L,C,T): \\ K \leftarrow_{\mathrm{R}} \mathsf{Dec}(\mathrm{SK},L,C) \\ \textit{if } K = \perp \textit{ or } \mathsf{pred}(K,T) = 0 \textit{ then return } \perp \\ \textit{else return } K \end{array}$$

A LKEM lkem *is said to be* IND-CCCA-secure *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{lkem},\mathsf{pred}}(\mathcal{A})$ is a negligible function in λ.*

Remark 3 (comparison with [30]). We point out the differences between our formulation and that in [30]. First, we consider a setting with labels. Second, in [30], the predicate is specified by the adversary via a circuit. Here, we consider a fixed predicate that takes an additional input T. To capture the prior formulation, the predicate would be circuit evaluation and T would be a circuit. Third, by fixing the predicate, we avoid having to explicitly consider plaintext uncertainty.

4.2 The TLS Labeled KEM

Following [32], we describe a labeled KEM which is extracted from the TLS protocol. Unlike [32] which stopped at analyzing (labeled) CCA-security of the ensuing scheme for the RSA mode of TLS, we show

how to derive SACCE security of TLS (for any mode) based on the IND-CCCA security of the labeled KEM. We then prove this IND-CCCA property to hold for the KEMs arising in various TLS modes, namely TLS-RSA, TLS-CCA, TLS-DH, and TLS-DHE. In Figure 4 we rewrite the TLS protocol from Figure 2 in terms of tlskem.

LKEMs from TLS. Given a generic TLS protocol parameterized by $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ (see Fig. 2) along with cryptographic components Kdf and PRF, we consider the LKEM tlskem with algorithms $(\mathsf{tls.Gen}, \mathsf{tls.Enc}, \mathsf{tls.Dec})$, in which $\mathsf{tls.Gen}(1^\lambda)$ is the same as KeyGen and the algorithms tls.Enc, tls.Dec are as below.

$\underline{\mathsf{tls.Enc}(\mathrm{PK}, \eta \| \mathrm{CERT}_S):}$
$(\mathrm{CRES}, \mathrm{PMS}) \leftarrow \mathsf{F}_C(\mathrm{PK});$
$\mathrm{MS} := \mathsf{Kdf}(\mathrm{PMS}, \eta);$
$\mathrm{UCFIN} := \mathsf{PRF}(\mathrm{MS}, 1 \| \eta \| \mathrm{CERT}_S \| \mathrm{CRES});$
$\mathrm{AKEY} := \mathsf{PRF}(\mathrm{MS}, 0 \| \eta);$
$\mathrm{USFIN} := \mathsf{PRF}(\mathrm{MS}, 2 \| \eta \| \mathrm{CERT}_S \| \mathrm{CRES} \| \mathrm{UCFIN});$
output $(\mathrm{CRES},\ \mathrm{AKEY} \| \mathrm{USFIN} \| \mathrm{UCFIN})$.

$\underline{\mathsf{tls.Dec}(\mathrm{SK}, \eta \| \mathrm{CERT}_S, \mathrm{CRES}):}$
$\mathrm{PMS} \leftarrow \mathsf{F}_S(\mathrm{SK}, \mathrm{CRES});$
if $\mathrm{PMS} = \perp$, set $\mathrm{PMS} \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$;
$\mathrm{MS} := \mathsf{Kdf}(\mathrm{PMS}, \eta);$
$\mathrm{UCFIN} := \mathsf{PRF}(\mathrm{MS}, 1 \| \eta \| \mathrm{CERT}_S \| \mathrm{CRES});$
$\mathrm{AKEY} := \mathsf{PRF}(\mathrm{MS}, 0 \| \eta);$
$\mathrm{USFIN} := \mathsf{PRF}(\mathrm{MS}, 2 \| \eta \| \mathrm{CERT}_S \| \mathrm{CRES} \| \mathrm{UCFIN});$
output $\mathrm{AKEY} \| \mathrm{USFIN} \| \mathrm{UCFIN}$.

In order to consider CCCA security we augment tlskem with the following predicate.

$\underline{\mathsf{tls.Pred}(\mathrm{AKEY} \| \mathrm{USFIN} \| \mathrm{UCFIN}, \mathrm{CFIN}):}$
$(\mathrm{ST}_e^S, \mathrm{ST}_d^S) \leftarrow \mathsf{stE.Init}(1^\lambda);$
check if $\mathrm{UCFIN} = \mathsf{stE.Dec}(\mathrm{CKEY}, \mathrm{H}_C, \mathrm{CFIN}, \mathrm{ST}_d^S)$.

Remark 4 (KEM key). Note that if were to define the KEM key as MS instead of $\mathrm{AKEY} \| \mathrm{USFIN} \| \mathrm{UCFIN}$, then the scheme derived from TLS-RSA would be insecure. We incorporate USFIN into the KEM key so that we may simulate the honest server's `Finished` messages in the proof of security, and we incorporate UCFIN into the KEM key so that tls.Pred can verify the validity of the encrypted client `Finished` message.

Remark 5 (client authentication). In the setting with client authentication in Section 8, we consider the same LKEM except we replace CERT_S with $\mathrm{CERT}_S \| \mathrm{CERT}_C$ throughout.

Remark 6 (randomizing PMS). The second step in tls.Dec "if $\mathrm{PMS} = \perp$, set $\mathrm{PMS} \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$" is important to mask decryption failures in the presence of a timing channel and thereby preventing Bleichenbacher-style attacks; from a functionality perspective, it only leads to a negligible difference from immediately outputting $\perp$ if $\mathrm{PMS} = \perp$.

4.3 SACCE Security: Proof of Theorem 1

We proceed to prove Theorem 1, as presented at the beginning of this section.

Correctness. If both parties π_i^C and π_j^S agree on the prefix $\mathrm{CREQ} \| \mathrm{SRES} \| \mathrm{CRES}$, then they must agree on PMS (by correctness of $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ as a KEM). This means they also agree on MS, η and AKEY. In addition, if both parties accept, then they output the same AKEY. Moreover, since both parties agree on SRES and thus both CERT_S and ID_S, C will output $\mathrm{PEER} = \mathrm{ID}_S$. This holds even if the adversary makes Reveal queries and mauls the `Finished` messages.

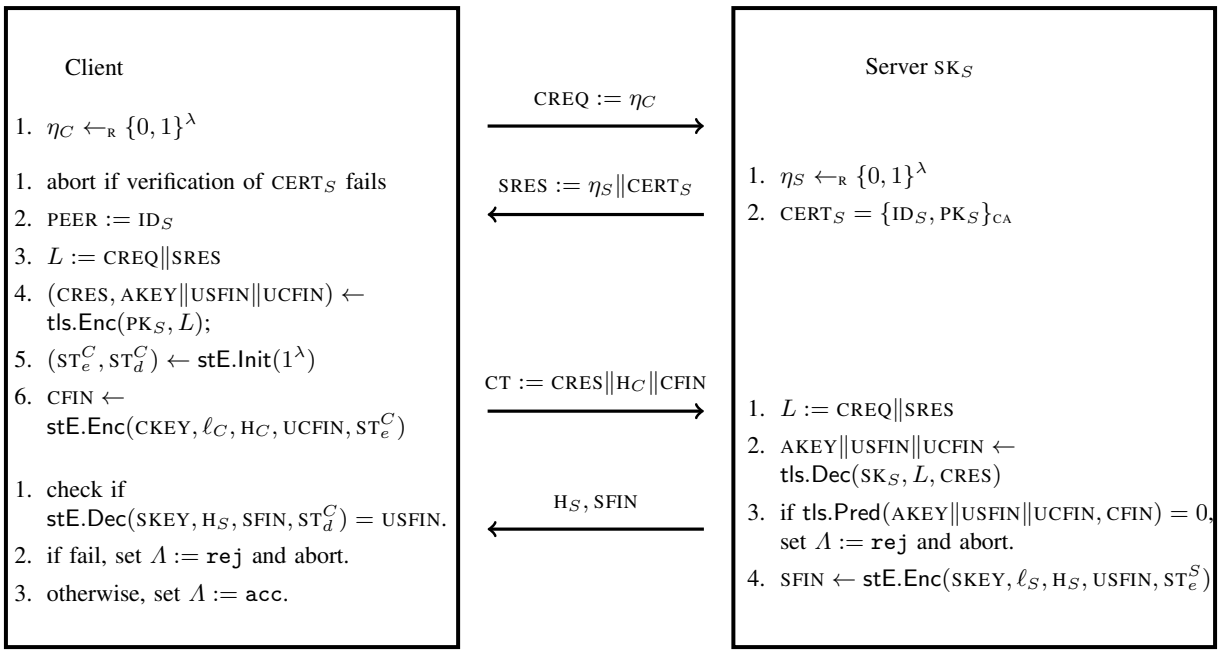

Fig. 4. Basic Generic TLS Handshake Protocol Parameterized by $\mathsf{tlskem} = (\mathsf{tls.Gen}, \mathsf{tls.Enc}, \mathsf{tls.Dec})$

High-level overview of SACCE security proof. The high-level idea for establishing server authentication and channel security is the same, and both rely on IND-CCCA security crucially. Let i^* denote the target client session in the selective SACCE game. We will embed the challenge ciphertext in the IND-CCCA security experiment sent by $\pi_{i^*}^C$, while using the restricted decryption oracle to simulate all the server responses. We will rely on IND-CCCA security to argue that the ciphertext sent by $\pi_{i^*}^C$ protects the privacy of $\text{AKEY}\|\text{USFIN}$; the privacy of AKEY gives us channel security, whereas the unpredictability of USFIN gives us server authentication. Moreover, $\text{AKEY}\|\text{USFIN}$ as computed by $\pi_{i^*}^C$ is "independent" of that in the other sessions. Note that IND-CCCA guarantees independence whenever the labels are different, and in TLS, the labels contains both the server's certificate and the server nonce. Roughly speaking, for server authentication, independence between sessions of different servers is guaranteed by the fact that the adversary cannot forge a certificate for an honest server, while for channel security, independence across the honest server sessions is guaranteed by the fact that all the nonces in these sessions are distinct with high probability. Once we switch $\text{AKEY}^*\|\text{USFIN}^*$ as computed by the honest client $\pi_{i^*}^C$ to a random value, we may rely on sLHAE security of stE to achieve channel security.

Server authentication. We begin with server authentication:

Lemma 2 (s-sacce-sa). *For any adversary $\mathcal{A}$, there exists adversaries $\mathcal{A}_1, \mathcal{B}$ such that*

$$\mathbf{Adv}_{\Pi}^{\text{s-sacce-sa}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{casig}}^{\text{sig}}(\mathcal{A}_1) + \mathbf{Adv}_{\mathsf{tlskem},\mathsf{tls.Pred}}^{\text{ind-ccca}}(\mathcal{B}) + (d_{max}^2 + 1) \cdot 2^{-\lambda}$$

Moreover, the running times of $\mathcal{A}_1, \mathcal{B}$ are roughly that of $\mathcal{A}$.

In this proof, we rely on the assumptions we make about the certificate authority (see Section 3.1), in particular, that it uses a secure signature scheme and that the adversary cannot register a public key of its choice for an honest server (otherwise, it can trivially break server authentication). At a high level, the proof proceeds as follows: we rely on IND-CCCA security to replace $(\text{AKEY}, \text{USFIN}, \text{UCFIN})$

computed by the target client session $\pi_{i^*}^C$ from real to random, upon which server authentication follows immediately from statistical unpredictability of a truly random USFIN.

Proof. Let i^* denote the target session in the selective SACCE game, and ID_S, PK_S denote the identity and public key of the honest server S. We use $\text{CREQ}^*, \text{SRES}^*, \text{CT}^*, \text{SFIN}^*$ to denote the messages sent or received by $\pi_{i^*}^C$. We proceed via a series of games. We use $\mathbf{Adv}_0, \mathbf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc.

Game 0. Real experiment.

Game 1. The challenger proceeds as before, but it aborts if any of the following holds:

- $(\text{ID}_S^*, \text{PK}_S^*) \neq (\text{ID}_S, \text{PK}_S)$; or
- the transcript at any server oracle π_j^S begins with $(\text{CREQ}^*, \text{SRES}^*, \text{CRES}^*)$;
- if the server nonces are not all distinct.

We claim that the first and second aborts do not affect the advantage of the adversary (unless the adversary manages to forge the CA's signature on on $(\text{ID}_S^*, \text{PK}_S^*)$, an event with probability bounded by $\mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1)$), and the third happens with probability $d_{\text{max}}^2 \cdot 2^{-\lambda}$, so

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 + \mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1) + d_{\text{max}}^2 \cdot 2^{-\lambda}$$

We consider each of the first two abort conditions separately:

- For the first condition, observe that if $(\text{ID}_S^*, \text{PK}_S^*) \neq (\text{ID}_S, \text{PK}_S)$, then either
  - $\text{ID}_S^* \neq \text{ID}_S$, and thus $\text{PEER} \neq \text{ID}_S$ and the adversary also aborts in Game 0; or
  - CERT_S^* fails verification and the adversary also aborts in Game 0; or
  - $\text{ID}_S^* = \text{ID}_S, \text{PK}_S^* \neq \text{PK}_S$ and CERT_S^* passes verification, in which case we have a signature forgery since the CA will not sign $(\text{ID}_S, \text{PK}_S^*)$.
- For the second, suppose the transcript at some server oracle π_j^S begins with $(\text{CREQ}^*, \text{SRES}^*, \text{CRES}^*)$. We consider two cases: if $\pi_{i^*}^C$ eventually reaches accept state, then it has a matching conversation to π_j^S, so $\mathcal{A}$ would not win in Game 0. If $\pi_{i^*}^C$ does not reach accept state, then $\mathcal{A}$ would not win in Game 0 either.

We note that if the challenger does not abort at the end of Game 1, then any matching conversation for $\pi_{i^*}^C$ must be unique, due to the server nonces being distinct.

Game 2. The challenger proceeds as before, but replaces the $(\text{AKEY}, \text{USFIN}, \text{UCFIN})$ computed by $\pi_{i^*}^C$ with a random $(\widetilde{\text{AKEY}}, \widetilde{\text{USFIN}}, \widetilde{\text{UCFIN}}) \leftarrow_{\text{R}} \{0,1\}^{4\lambda}$. That is,

- parse $\widetilde{\text{AKEY}}$ as $\widetilde{\text{CKEY}} \| \widetilde{\text{SKEY}}$;
- $\pi_{i^*}^C$ computes $\text{CT}^* := \text{CRES}^* \| \text{H}_C^* \| \text{CFIN}^*$ at $\pi_{i^*}^C$ by first computing CRES^*, and then using $\widetilde{\text{CKEY}}$ to encrypt $\widetilde{\text{UCFIN}}$.
- $\pi_{i^*}^C$ decrypts $\text{H}_S^* \| \text{SFIN}^*$ using $\widetilde{\text{SKEY}}$ and checks equality with $\widetilde{\text{USFIN}}$.
- the challenger responds to $\mathsf{Reveal}(\pi_{i^*}^C)$ with $\widetilde{\text{AKEY}}$.

By IND-CCCA security of tlskem, we have (as shown below in Lemma 3):

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B})$$

Now, we have $\mathbf{Adv}_2 \leq 2^{-\lambda}$, because $\widetilde{\text{USFIN}}$ is truly random from the perspective of $\mathcal{A}$. Putting everything together, we obtain the bound on $\mathbf{Adv}^{\text{s-sacce-sa}}_{\Pi}(\mathcal{A})$ as claimed. □

Lemma 3 (ccca in s-acce-sa). *There exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B})$$

Proof. We construct an adversary $\mathcal{B}$ for the IND-CCCA security game of tlskem, such that if the challenge bit b is 0, $\mathcal{B}$ simulates Game 1, and if b equals 1, $\mathcal{B}$ simulates Game 2. (We neglect all the scenarios where the challenger aborts in Game 1.) As before, we use $\mathrm{CREQ}^*, \mathrm{SRES}^*, \mathrm{CT}^*, \mathrm{USFIN}^*$ to denote the messages sent or received by $\pi^C_{i^*}$, and we use $(\mathrm{CRES}^*, \widetilde{\mathrm{AKEY}}\|\widetilde{\mathrm{USFIN}}\|\widetilde{\mathrm{UCFIN}})$ to denote the challenge ciphertext and KEM key in the IND-CCCA experiment.

- $\mathcal{B}$ uses PK from the IND-CCCA security game as PK_S. $\mathcal{B}$ also gets ID_S and prepares CERT_S by simulating the CA.
- $\mathcal{B}$ simulates $\pi^C_{i^*}$ as follows:
  - pick $\mathrm{CREQ}^* := \eta^*_C \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$;
  - when $\mathcal{A}$ sends $\mathrm{SRES}^* := (\eta^*_S, \mathrm{CERT}^*_S)$ to $\pi^C_{i^*}$, do the following:
    1. abort if $(\mathrm{ID}^*_S, \mathrm{PK}^*_S) \neq (\mathrm{ID}_S, \mathrm{PK}_S)$;
    2. send $L^* := \mathrm{CREQ}^*\|\mathrm{SRES}^*$ to the challenger in the IND-CCCA security game (by uniqueness of η^*_S amongst server nonces, this is a valid L^*);
    3. parse the response as $(\mathrm{CRES}^*, \widetilde{\mathrm{AKEY}}\|\widetilde{\mathrm{USFIN}}\|\widetilde{\mathrm{UCFIN}})$ and parse $\widetilde{\mathrm{AKEY}}$ as $\widetilde{\mathrm{CKEY}}\|\widetilde{\mathrm{SKEY}}$;
    4. use $\widetilde{\mathrm{CKEY}}$ to encrypt $\widetilde{\mathrm{UCFIN}}$ to obtain $\mathrm{H}^*_C\|\mathrm{CFIN}^*$ and send $\mathrm{CT}^* := \mathrm{CRES}^*\|\mathrm{H}^*_C\|\mathrm{CFIN}^*$ to $\mathcal{A}$ as the response from $\pi^C_{i^*}$;
  - $\pi^C_{i^*}$ decrypts $\mathrm{H}^*_S\|\mathrm{SFIN}^*$ using $\widetilde{\mathrm{SKEY}}$ and checks equality with $\widetilde{\mathrm{USFIN}}$;
  - the challenger responds to $\mathsf{Reveal}(\pi^C_{i^*})$ with $\widetilde{\mathrm{AKEY}}$.

  Observe if the challenge bit b in the IND-CCCA security game is 0, then $\widetilde{\mathrm{USFIN}}\|\widetilde{\mathrm{AKEY}}$ is computed as in Game 1, and if b is 1, then $\widetilde{\mathrm{USFIN}}\|\widetilde{\mathrm{AKEY}}$ is truly random as in Game 2.
- $\mathcal{B}$ simulates all π^C_i for $i \neq i^*$ as in Game 1. This means $\mathcal{B}$ can answer any Reveal query for π^C_i too.
- $\mathcal{B}$ simulates π^S_j for all j as follows: compute SRES as in Game 1; on input CT, set $L := \mathrm{CREQ}\|\mathrm{SRES}$ and compute $\mathrm{AKEY}\|\mathrm{USFIN}$ as follows:
  - if $\mathcal{B}$ has not sent L^* in the IND-CCCA security game or $(L, \mathrm{CRES}) \neq (L^*, \mathrm{CRES}^*)$, query $\mathsf{CDec}(\mathrm{SK}, \cdot)$ on input (L, CT) to obtain $\mathrm{AKEY}\|\mathrm{USFIN}$;
  - if $\mathcal{B}$ has sent L^* in the IND-CCCA security game and $(L, \mathrm{CRES}) = (L^*, \mathrm{CRES}^*)$, then we may simply abort as in Game 1.

  Note that $\mathcal{B}$ learns AKEY, so it can answer any Reveal query for π^S_j too.

The claim follows readily. □

Channel security. We complete the proof by establishing channel security:

Lemma 4 (s-sacce-ae). *For any adversary $\mathcal{A}$, there exists adversaries $\mathcal{B}, \mathcal{A}_2$ and $\mathcal{A}_3$ such that*

$$\mathbf{Adv}^{\mathrm{s-sacce-ae}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B}) + \mathbf{Adv}^{\text{lh-st-ae}}_{\mathsf{stE}}(\mathcal{A}_2) + \mathbf{Adv}^{\text{lh-st-ae}}_{\mathsf{stE}}(\mathcal{A}_3) + d_{max} \cdot 2^{-\lambda}$$

where d_{max} is an upper bound on the number of sessions invoked by $\mathcal{A}$ at any party. Moreover, the running times of $\mathcal{B}, \mathcal{A}_2$ and $\mathcal{A}_3$ are roughly that of $\mathcal{A}$.

At a high level, the proof proceeds as follows: we rely on IND-CCCA security to replace $(\text{AKEY}, \text{USFIN}, \text{UCFIN})$ computed by the target client session $\pi^C_{i^*}$ from real to random, upon which channel security follows from sLHAE security with a truly random AKEY. The most delicate step in whole SACCE security proof lies in the transition from Game 1 to Game 2 in this lemma, where we switch $(\text{AKEY}, \text{USFIN}, \text{UCFIN})$ from real to random, so that we may exploit non-malleability of stE to rule out malleability attacks on CFIN.

Proof. Let i^* and j^* denote the target sessions in the selective SACCE game, and ID_S, PK_S denote the identity and public key of the honest server S. We use $\text{CREQ}^*, \text{SRES}^*, \text{CT}^*, \text{SFIN}^*$ to denote the messages sent or received by $\pi^C_{i^*}$. Note that the matching requirement in channel security stipulates that the transcripts at both $\pi^C_{i^*}$ and $\pi^S_{j^*}$ must begin with the same $(\text{CREQ}^*, \text{SRES}^*, \text{CRES}^*)$. We proceed via a series of games. We use $\mathbf{Adv}_0, \mathbf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc.

Game 0. Real experiment. Henceforth, we neglect the cases where b' (as output by $\mathcal{A}$) is $\perp$.

Game 1. The challenger in this game proceeds as before, but it aborts if any of the following holds:

- any server nonce for $j \neq j^*$ matches that for $\pi^S_{j^*}$; or
- the transcript at server oracle $\pi^S_{j^*}$ does not begin with $(\text{CREQ}^*, \text{SRES}^*, \text{CRES}^*)$.

Thus,

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 - d_{\max} \cdot 2^{-\lambda}$$

In particular, if the challenger does not abort, then $\pi^C_{i^*}$ has a unique matching conversation with $\pi^S_{j^*}$.

Game 2. The challenger proceeds as before, except it replaces $(\text{AKEY}, \text{USFIN}, \text{UCFIN})$ computed by $\pi^C_{i^*}$ with a random $(\widetilde{\text{AKEY}}, \widetilde{\text{USFIN}}, \widetilde{\text{UCFIN}}) \leftarrow_{\text{R}} \{0,1\}^{4\lambda}$. Specifically:

- parse $\widetilde{\text{AKEY}}$ as $\widetilde{\text{CKEY}} \| \widetilde{\text{SKEY}}$;
- $\pi^C_{i^*}$ computes CT^* as follows:
  1. generate CRES^* using $\mathsf{tls.Enc}$ as in Game 1;
  2. compute $\text{H}^*_C \| \text{CFIN}^*$ by encrypting $\widetilde{\text{UCFIN}}$ with $\widetilde{\text{CKEY}}$;
  3. set $\text{CT}^* := \text{CRES}^* \| \text{H}^*_C \| \text{CFIN}^*$ as in Game 1.
- $\pi^S_{j^*}$ computes the server $\mathtt{Finished}$ message on input $\text{CT} := \text{CRES} \| \text{H}_C \| \text{CFIN}$ as follows:
  1. if $\text{CT} = \text{CT}^*$: sets $\Lambda := \mathtt{acc}$ and responds with the encryption of $\widetilde{\text{USFIN}}$ under $\widetilde{\text{SKEY}}$;
  2. if $\text{CRES} = \text{CRES}^*$ but $\text{H}_C \| \text{CFIN} \neq \text{H}^*_C \| \text{CFIN}^*$: if decrypting $\text{H}_C \| \text{CFIN}$ with $\widetilde{\text{CKEY}}$ yields $\widetilde{\text{UCFIN}}$, sets $\Lambda := \mathtt{acc}$ and responds with the encryption of $\widetilde{\text{USFIN}}$ under $\widetilde{\text{SKEY}}$, and otherwise, sets $\Gamma := \mathtt{rej}$ and aborts;
  3. if $\text{CRES} \neq \text{CRES}^*$: aborts as in Game 1.
- $\pi^C_{i^*}$ verifies $\text{H}^*_S \| \text{SFIN}^*$ it receives from $\mathcal{A}$ by decrypting with $\widetilde{\text{SKEY}}$ and checking equality with $\widetilde{\text{USFIN}}$, and aborts if verification fails.
- answer $\mathsf{Encrypt}(\pi^C_{i^*}, \ldots)$ and $\mathsf{Decrypt}(\pi^C_{i^*}, \ldots)$ queries using $\widetilde{\text{CKEY}}$.
- answer $\mathsf{Encrypt}(\pi^S_{j^*}, \ldots)$ and $\mathsf{Decrypt}(\pi^S_{j^*}, \ldots)$ queries using $\widetilde{\text{SKEY}}$.

By IND-CCCA security of tlskem, we have (as shown below in Lemma 5):

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{ind-ccca}}_{\mathsf{tlskem}, \mathsf{tls.Pred}}(\mathcal{B})$$

Game 3. The challenger proceeds as before, except we modify $\pi^S_{j^*}$ as follows:

- $\pi^S_{j^*}$ computes the server $\mathtt{Finished}$ message on input $\text{CT} := \text{CRES} \| \text{H}_C \| \text{CFIN}$ as follows:

1. if $\text{CT} = \text{CT}^*$: sets $\Lambda := \texttt{acc}$ and responds with the encryption of $\widetilde{\text{USFIN}}$ under $\widetilde{\text{SKEY}}$ as in Game 2;
2. if $\text{CRES} = \text{CRES}^*$ but $\text{H}_C \| \text{CFIN} \neq \text{H}^*_C \| \text{CFIN}^*$: sets $\Gamma := \texttt{rej}$ and abort (this is the only difference from Game 2);
3. if $\text{CRES} \neq \text{CRES}^*$: aborts as in Game 2.

It is straight-forward to construct $\mathcal{A}_2$ so that

$$\mathbf{Adv}_2 \leq \mathbf{Adv}_3 + \mathbf{Adv}^{\text{lh-st-ae}}_{\mathsf{stE}}(\mathcal{A}_2)$$

We use the fact that in Game 2, the key material $\widetilde{\text{CKEY}}$ used by $\pi^C_{i^*}$ and $\pi^S_{j^*}$ is chosen uniformly at random. Specifically, $\mathcal{A}_2$ is given access to an encryption oracle Enc and a decryption oracle Dec where the key corresponds to $\widetilde{\text{CKEY}}$, and proceeds as follows:

- simulates all oracles different from $\pi^C_{i^*}$ and $\pi^S_{j^*}$ as in Game 2;
- simulates all messages up to CRES^* between $\pi^C_{i^*}$ and $\pi^S_{j^*}$ and the computation of $\widetilde{\text{UCFIN}}$ as in Game 2;
- picks $\widetilde{\text{SKEY}} \leftarrow_{\text{R}} \{0,1\}^\lambda$ and $\widetilde{\text{USFIN}} \leftarrow_{\text{R}} \{0,1\}^\lambda$ as in Game 2;
- simulates $\pi^C_{i^*}$'s encryption of $(\text{H}^*_C, \widetilde{\text{UCFIN}})$ by querying Enc on input $(\text{H}^*_C, \widetilde{\text{UCFIN}}, \widetilde{\text{UCFIN}})$;
- simulates $\pi^S_{j^*}$'s decryption of $\text{H}_C \| \text{CFIN}$ (where $\text{H}_C \| \text{CFIN} \neq \text{H}^*_C \| \text{CFIN}^*$) by querying Dec on input $(\text{H}_C, \text{CFIN})$;
- forwards all subsequent $\mathsf{Encrypt}(\pi^C_{i^*}, \cdot)$ queries to Enc;
- forwards all subsequent $\mathsf{Decrypt}(\pi^C_{i^*}, \cdot)$ queries to Dec.
- answers all subsequent $\mathsf{Encrypt}(\pi^S_{j^*}, \cdot)$ and $\mathsf{Decrypt}(\pi^S_{j^*}, \cdot)$ queries using $\widetilde{\text{SKEY}}$.

Let b denote the challenge bit in the sLHAE security experiment. Then, when $b = 1$, we simulate Game 2, and when $b = 0$, we simulate Game 3.

Now, we obtain an upper bound on $\mathbf{Adv}_3$, by using the fact that in Game 3, the key material $\widetilde{\text{CKEY}} \| \widetilde{\text{SKEY}}$ used by $\pi^C_{i^*}$ and $\pi^S_{j^*}$ is chosen uniformly at random. That is, we can construct $\mathcal{A}_3$ (as in Game 6 of [31, Lemma 5]) so that

$$\mathbf{Adv}_3 \leq 1/2 + \mathbf{Adv}^{\text{lh-st-ae}}_{\mathsf{stE}}(\mathcal{A}_3)$$

Specifically, $\mathcal{A}_3$ is given access to an encryption oracle Enc and a decryption oracle Dec where the challenge bit b correponds to $b^C_{i^*}$ and the key corresponds to $\widetilde{\text{CKEY}}$, and proceeds as follows:

- simulates all oracles different from $\pi^C_{i^*}$ and $\pi^S_{j^*}$ as in Game 3;
- simulates all messages up to CRES^* between $\pi^C_{i^*}$ and $\pi^S_{j^*}$ and the computation of $\widetilde{\text{UCFIN}}$ as in Game 3;
- picks $\widetilde{\text{SKEY}} \leftarrow_{\text{R}} \{0,1\}^\lambda$ and $\widetilde{\text{USFIN}} \leftarrow_{\text{R}} \{0,1\}^\lambda$ as in Game 3;
- simulates $\pi^C_{i^*}$'s encryption of $(\text{H}^*_C, \widetilde{\text{UCFIN}})$ by querying Enc on input $(\text{H}^*_C, \widetilde{\text{UCFIN}}, \widetilde{\text{UCFIN}})$;
- forwards all subsequent $\mathsf{Encrypt}(\pi^C_{i^*}, \cdot)$ queries to Enc;
- forwards all subsequent $\mathsf{Decrypt}(\pi^C_{i^*}, \cdot)$ queries to Dec.
- answers all subsequent $\mathsf{Encrypt}(\pi^S_{j^*}, \cdot)$ and $\mathsf{Decrypt}(\pi^S_{j^*}, \cdot)$ queries using $\widetilde{\text{SKEY}}$.

Putting everything together, we obtain the bound on $\mathbf{Adv}^{\text{s-sacce-ae}}_{\Pi}(\mathcal{A})$ as claimed. □

Lemma 5 (ccca in s-sacce-ae). *There exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B})$$

Proof. We construct an adversary $\mathcal{B}$ for the IND-CCCA security game of tr.tlskem, such that if the challenge bit b is 0, $\mathcal{B}$ simulates Game 1, and if b equals 1, $\mathcal{B}$ simulates Game 2. (We neglect all the scenarios where the challenger aborts in Game 1.) As before, we use CREQ^*, SRES^*, CT^*, SFIN^* to denote the messages sent or received by $\pi^C_{i^*}$, and we use $(\text{CRES}^*, \widetilde{\text{AKEY}}\|\widetilde{\text{USFIN}}\|\widetilde{\text{UCFIN}})$ to denote the challenge ciphertext and KEM key in the IND-CCCA experiment.

- $\mathcal{B}$ uses PK from the IND-CCCA security game as PK_S. $\mathcal{B}$ also gets ID_S and prepares CERT_S by simulating the CA.
- $\mathcal{B}$ simulates $\pi^C_{i^*}$ and $\pi^S_{j^*}$ using $\widetilde{\text{AKEY}}\|\widetilde{\text{USFIN}}$ in Game 2 as follows:
  - $\pi^C_{i^*}$ computes CT^* as follows:
    1. send $L^* := \text{CREQ}^*\|\text{SRES}^*$ to the challenger in the IND-CCCA security game (by uniqueness of η^*_S amongst server nonces, this is a valid L^*);
    2. obtain $(\text{CRES}^*, \widetilde{\text{AKEY}}\|\widetilde{\text{USFIN}}\|\widetilde{\text{UCFIN}})$ from the challenger in the IND-CCCA experiment;
    3. parse $\widetilde{\text{AKEY}}$ as $\widetilde{\text{CKEY}}\|\widetilde{\text{SKEY}}$;
    4. compute $\text{H}^*_C\|\text{CFIN}^*$ by encrypting $\widetilde{\text{UCFIN}}$ with $\widetilde{\text{CKEY}}$;
    5. set $\text{CT}^* := \text{CRES}^*\|\text{H}^*_C\|\text{CFIN}^*$ in Game 1.
  - $\pi^S_{j^*}$ computes the server `Finished` message on input $\text{CT} := \text{CRES}\|\text{H}_C\|\text{CFIN}$ as follows:
    1. if $\text{CRES} = \text{CRES}^*$, then $\pi^S_{j^*}$ decrypts $\text{H}_C\|\text{CFIN}$ with $\widetilde{\text{CKEY}}$, checks equality with $\widetilde{\text{UCFIN}}$, and if equal, sets $\Lambda := \texttt{acc}$ and responds with the encryption of $\widetilde{\text{USFIN}}$ under $\widetilde{\text{SKEY}}$ and otherwise, sets $\Gamma := \texttt{rej}$ and abort;
    2. if $\text{CRES} \neq \text{CRES}^*$, then abort as in Game 1.

    We need to verify that if $b = 0$, then $\pi^S_{j^*}$ is behaving exactly as in Game 1. The case $\text{CRES} \neq \text{CRES}^*$, so we focus on $\text{CRES} = \text{CRES}^*$. Let AKEY, UCFIN, USFIN denote the values computed by an honest $\pi^S_{j^*}$ on input CT in Game 1. Now, $\text{CREQ}\|\text{SRES}\|\text{CRES}$ completely determines an accepting AKEY, UCFIN, USFIN. Since $\text{CREQ}\|\text{SRES}\|\text{CRES} = \text{CREQ}^*\|\text{SRES}^*\|\text{CRES}^*$ and $b = 0$, we have that if CT is accepting, then

    $$(\text{AKEY}, \text{UCFIN}, \text{USFIN}) = (\widetilde{\text{AKEY}}, \widetilde{\text{UCFIN}}, \widetilde{\text{USFIN}})$$

    Therefore, we may decrypt with $\widetilde{\text{AKEY}}$, check equality with $\widetilde{\text{UCFIN}}$, and if equality holds, respond with $\widetilde{\text{USFIN}}$.
  - $\pi^C_{i^*}$ verifies $\text{H}^*_S\|\text{SFIN}^*$ it receives from $\mathcal{A}$ by decrypting with $\widetilde{\text{SKEY}}$ and checking equality with $\widetilde{\text{USFIN}}$, and aborts if verification fails.
  - answer $\mathsf{Encrypt}(\pi^C_{i^*}, \ldots)$ and $\mathsf{Decrypt}(\pi^C_{i^*}, \ldots)$ queries using $\widetilde{\text{CKEY}}$.
  - answer $\mathsf{Encrypt}(\pi^S_{j^*}, \ldots)$ and $\mathsf{Decrypt}(\pi^S_{j^*}, \ldots)$ queries using $\widetilde{\text{SKEY}}$.
- $\mathcal{B}$ simulates all π^C_i for $i \neq i^*$ as in Game 1.
- $\mathcal{B}$ simulates π^S_j for all $j \neq j^*$ as follows: compute η_S as in Game 1; and on input CT, compute AKEY and USFIN as follows:
  - query $\mathsf{CDec}(\text{SK}, \cdot)$ on input $(\text{CRES}\|\text{SRES}, \text{CT})$, and use the response as $\text{AKEY}\|\text{USFIN}$ (this is a valid query by uniqueness of η^*_S).

  Note that $\mathcal{B}$ can compute AKEY, so it can answer any Reveal query for π^S_j too.

The claim follows readily. □

5 TLS-RSA: Instantiations from OW-PCA

Here, we show that if the underlying KEM $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ is OW-PCA secure, then the tlskem scheme in Section 4.2 is IND-CCCA-secure in the random oracle model. Hence, by Theorem 1, the corresponding TLS scheme is SACCE-secure. In Section 5.2 we apply this result to show the security of TLS-RSA.

OW-PCA for KEM [41]. For a stateful adversary $\mathcal{A}$ and a KEM kem with algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, we define the advantage function

$$\mathbf{Adv}^{\mathrm{ow-pca}}_{\mathsf{kem}}(\mathcal{A}) := \Pr\left[K' = K^* : \begin{array}{r}(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda);\\ (\psi^*, K^*) \leftarrow \mathsf{Enc}(\mathrm{PK});\\ K' \leftarrow \mathcal{A}^{\mathsf{PCA}(\mathrm{SK},\cdot,\cdot)}(\psi^*)\end{array}\right]$$

where $\mathsf{PCA}(\mathrm{SK}, \cdot, \cdot)$ is the oracle that takes as input (K, ψ) with $K \neq \perp$ and outputs 1 if $\mathsf{Dec}(\mathrm{SK}, \psi) = K$ and 0 otherwise. An encryption scheme is said to be *one-way against plaintext checking attacks* (OW-PCA) if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathrm{ow-pca}}_{\mathsf{kem}}(\mathcal{A})$ is a negligible function in λ.

5.1 IND-CCCA from OW-PCA

The following lemma is similar to that in [32, Theorem 3], with some significant differences: (i) the KEM key in [32, Theorem 3] is AKEY and the ciphertext is CRES$\|$UCFIN, whereas our KEM key is AKEY$\|$USFIN$\|$UCFIN and the ciphertext is simply CRES; (ii) [32] models PRF (referred to as h_s and h_z therein) also as a random oracle; (iii) [32] proves (labeled) IND-CCA security and does not consider encryption of UCFIN.

Lemma 6 (IND-CCCA from OW-PCA). *If* $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ *is OW-PCA secure,* PRF *is a pseudorandom function, and we model* $\mathsf{Kdf}()$ *as a random oracle, then the LKEM* tlskem *with predicate* tls.Pred *(in Section 4.2) is IND-CCCA secure in the random oracle model (c.f. Section 4.1). That is, for any adversary* $\mathcal{A}$ *that makes at most* Q *decryption queries, there exists adversaries* $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_4$ *such that*

$$\mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{lkem,pred}}(\mathcal{A}) \leq Q \cdot (\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A}_1) + 2^{-\lambda}) + \mathbf{Adv}^{\mathrm{ow-pca}}_{\mathsf{kem}}(\mathcal{A}_2) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A}_4).$$

Moreover, the running times of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_4$ *are roughly that of* $\mathcal{A}$.

We begin with a high-level overview of the proof. The main challenge lies in simulating the CDec oracle without knowing SK. Let $(\eta^* \| \mathrm{CERT}^*_S, \mathrm{CRES}^*)$ denote the challenge ciphertext and let $(\eta \| \mathrm{CERT}_S, \mathrm{CRES}, \mathrm{CFIN})$ denote a query to the CDec oracle. Let PMS and PMS* denote $\mathsf{F}_S(\mathrm{SK}, \mathrm{CRES})$ and $\mathsf{F}_S(\mathrm{SK}, \mathrm{CRES}^*)$ respectively. Recall that the adversary also gets either K_0 or K_1 where K_0 is the real key and K_1 is a random key. We proceed via a case analysis:

- if the adversary has queried $\mathsf{Kdf}()$ at (PMS, η), then we can easily answer the decryption query;
- if PMS $=\perp$, then we output $\perp$.
- if the adversary has not queried $\mathsf{Kdf}()$ at (PMS, η) and $(\mathrm{PMS}, \eta) \neq (\mathrm{PMS}^*, \eta^*)$, then $\mathsf{Kdf}(\mathrm{PMS}, \eta)$ is statistically random from the adversary's view-point and the adversary will not be able to provide a consistent CFIN (and therefore we can always output $\perp$ for such queries);
- if $(\mathrm{PMS}, \eta) = (\mathrm{PMS}^*, \eta^*)$, then we consider two sub-cases (note that it is possible that PMS $=$ PMS* while CRES $\neq$ CRES*). If the adversary has queried $\mathsf{Kdf}()$ at (PMS, η), then it breaks one-wayness.

If it has not, we cannot argue as before that $\mathsf{Kdf}(\text{PMS}, \eta)$ is statistically random from the adversary's view-point $\mathsf{Kdf}(\text{PMS}, \eta) = \mathsf{Kdf}(\text{PMS}^*, \eta^*)$; this is because K_0 leaks information about $\text{MS}^* := \mathsf{Kdf}(\text{PMS}^*, \eta^*)$. Nonetheless, we can still invoke pseudo-randomness of $\mathsf{PRF}(\text{MS}^*, \cdot)$ to argue that the adversary will not be able to provide a consistent CFIN.

Note that by observing the $\mathsf{Kdf}()$ oracle queries and using the $\mathsf{PCA}()$ oracle, we can identify queries that fall into first or the last cases, but we cannot distinguish between queries in the second and third case. Fortunately, we can provide the same answer $\perp$ for queries in both the second and third case.

Proof. We proceed via a series of games. We start with Game 0, where the challenger proceeds as in the real IND-CCCA game (i.e, K_0 is a real key and K_1 is a random key) and end up with a game where both K_0 and K_1 are chosen uniformly at random. We use $\mathbf{Adv}_0, \mathbf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc. Let $\text{PMS}^*, \text{CRES}^*, \text{MS}^*, \ldots$ denote the values used to compute the challenge ciphertext in the IND-CCCA experiment. We assume that the adversary $\mathcal{A}$ makes at most Q queries to the decryption oracle.

Game 0. Real experiment.

Game 1. For $i = 1, 2, \ldots, Q$, we simulate the response to the i'th decryption query $(\eta \| \text{CERT}_S, \text{CRES}, \text{CFIN})$ to $\mathsf{CDec}(\text{SK}, \cdot, \cdot, \cdot)$ as follows:

1. abort if $(\eta \| \text{CERT}_S, \text{CRES})$ equals (L^*, CRES^*) as in Game 0;
2. examine the prefix PMS' of each query (PMS', η') to $\mathsf{Kdf}()$ and check whether $\mathsf{F}_S(\text{SK}, \text{CRES})$ equals PMS' querying $(\text{PMS}', \text{CRES})$ to $\mathsf{PCA}(\text{SK}, \cdot)$ oracle. If we find a match, then we know PMS and can simulate the decryption oracle perfectly. If we do not find a match, respond with $\perp$.

We claim that there exists an adversary $\mathcal{A}_1$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 + Q \cdot (\mathbf{Adv}^{\text{prf}}_{\mathsf{PRF}}(\mathcal{A}_1) + 2^{-\lambda})$$

It suffices to show that for each i, if we do not find a match (upon which we respond with $\perp$), then we simulate the i'th decryption query correctly except with probability $\mathbf{Adv}^{\text{prf}}_{\mathsf{PRF}}(\mathcal{A}_1) + 2^{-\lambda}$. Roughly speaking, the reason is that UCFIN is statistically unpredictable from the view-point of $\mathcal{A}$. Fix a query $(\eta \| \text{CERT}_S, \text{CRES}, \text{CFIN})$ to CDec. We write $\text{PMS} := \mathsf{F}_S(\text{SK}, \text{CRES})$ and $\text{MS} := \mathsf{PRF}(\text{PMS}, \eta)$ (where in MS, we use a random PMS if $\text{PMS} = \perp$) and proceed via a case analysis:

– $(\text{PMS}, \eta) = (\text{PMS}^*, \eta^*)$: here, $\text{MS} = \text{MS}^*$, and the real key K_0 gives away information[8] about

$$\mathsf{PRF}(\text{MS}, 0\|\eta), \mathsf{PRF}(\text{MS}, 1\|L^*\|\text{CRES}^*), \mathsf{PRF}(\text{MS}, 2\|L^*\|\text{CRES}^*\|\text{UCFIN}^*).$$

By pseudorandomness of $\mathsf{PRF}(\text{MS}, \cdot)$, the value $\mathsf{PRF}(\text{MS}, 1\|L\|\text{CRES})$ remains unpredictable from the view-point of $\mathcal{A}$, since $(L\|\text{CRES}) \neq (L^*\|\text{CRES}^*)$. Therefore, $\mathcal{A}$ guesses UCFIN correctly with negligible probability. Formally, $\mathcal{A}_1$ will decrypt the $\text{H}_C\|\text{CFIN}$ provided by $\mathcal{A}$ using $\mathsf{PRF}(\text{MS}, 0\|\eta)$ and use the decrypted value as its guess for UCFIN.

– $\text{PMS} = \perp$ or $(\text{PMS}, \eta) \neq (\text{PMS}^*, \eta^*)$: here, $\text{MS} = \mathsf{Kdf}(\text{PMS}, \eta)$ is truly random from the view-point of $\mathcal{A}$.[9] Now, by pseudo-randomness, we may replace $\mathsf{PRF}(\text{MS}, \cdot)$ with a truly random function (we also use the same random function for all subsequent decryption queries with the same (PMS, η)). At this point, note that $\mathcal{A}$ guesses UCFIN correctly with probability at most $2^{-\lambda}$. Formally, $\mathcal{A}_1$ will also decrypt the $\text{H}_C\|\text{CFIN}$ provided by $\mathcal{A}$ using $\mathsf{PRF}(\text{MS}, 0\|\eta)$ and use the decrypted value as its guess for UCFIN.

[8] Here, we exploit the fact that we defined the KEM key in tlskem to be $\text{AKEY}\|\text{USFIN}\|\text{UCFIN}$ and not PMS or MS.

[9] Note that if tls.Dec simply outputs $\perp$ when $\text{PMS} = \perp$, then CDec will also output $\perp$, and the case $\text{PMS} = \perp$ follows trivially.

Game 2. We examine the prefix of each query $(\text{PMS}', \cdot)$ to $\mathsf{Kdf}()$ and check whether $\mathsf{F}_S(\text{SK}, \text{CRES}^*)$ equals PMS' by querying $(\text{PMS}', \text{CRES}^*)$ to $\mathsf{PCA}(\text{SK}, \cdot)$ oracle. We abort if we find such a match. We claim that there exists an adversary $\mathcal{A}_2$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{ow}-\text{pca}}_{\mathsf{kem}}(\mathcal{A}_2)$$

The reduction is straight-forward, since in Game 1, we answer all decryption oracles using only the $\mathsf{PCA}(\text{SK}, \cdot, \cdot)$ oracle. More precisely, $\mathcal{A}_2$ embeds the challenge in the $\mathbf{Adv}^{\text{ow}-\text{pca}}_{\mathsf{kem}}$ security game into CRES^*.

Game 3. We replace $\text{MS}^* := \mathsf{Kdf}(\text{PMS}^*, \eta^*)$ with $\text{MS}^* \leftarrow_{\text{R}} \{0,1\}^\lambda$ in the computation of UCFIN^*, AKEY^* and USFIN^* for K_0. We claim that

$$\mathbf{Adv}_2 = \mathbf{Adv}_3$$

This is because in Game 2, we abort whenever the adversary queries $\mathsf{Kdf}()$ on prefix PMS^*. Therefore, if we do not abort, then $\mathsf{Kdf}(\text{PMS}^*, \eta^*)$ must be truly random from the view-point of the adversary.

Game 4. We replace $\mathsf{PRF}(\text{MS}^*, \cdot)$ with a truly random function. It follows immediately from pseudo-randomness, that there exists an adversary $\mathcal{A}_4$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_3 \leq \mathbf{Adv}_4 + \mathbf{Adv}^{\text{prf}}_{\mathsf{PRF}}(\mathcal{A}_4)$$

Observe that in Game 4, the three values UCFIN^*, AKEY^*, USFIN^* are independent and uniformly random strings. That is, $K_0 := \text{AKEY}^* \| \text{USFIN}^* \| \text{UCFIN}^*$ has the uniform distribution, which is the same as that of K_1. Therefore, the view of the adversary is statistically independent of the challenge bit b. Therefore, $\mathbf{Adv}_4 = 0$. The claim follows readily. □

5.2 Implications for TLS-RSA

TLS-RSA KEM. The definition of the TLS-RSA KEM follows from the RSA PKCS #1v1.5 standard [33] adopted in TLS. Building on [32] we abstract the specification with parameters $\lambda_0 = \Theta(\lambda)$, $\lambda_1 = \Theta(\lambda)$ with $\lambda_0 \leq \lambda_1 - 88$ as follows:

- $\mathsf{KeyGen}(1^\lambda)$ is standard RSA key generation that outputs $(\text{PK}, \text{SK}) := ((N, e), d)$ where $de = 1 \pmod{\phi(N)}$ and N has λ_1 bits, where we assume that λ_1 is a multiple of 8.
- $\mathsf{F}_C : \{0,1\}^{\lambda_0} \to \mathbb{Z}^*_N$ takes as input λ_0-bit r, picks a padding $P \leftarrow_{\text{R}} \{0,1\}^{\lambda_1 - \lambda_0 - 24}$ at random (subject to none of the bytes of P being 00), sets $x := 00 \| 02 \| P \| 00 \| r$ (where $00, 02$ are byte encodings), and outputs $y := x^e \pmod{N}$.
- $\mathsf{F}_S : \mathbb{Z}^*_N \to \{0,1\}^{\lambda_0}$ takes as input y, and attempts to parse $y^d \pmod{N}$ as a byte sequence of the form $00 \| 02 \| P \| 00 \| r$ where P contains no zero bytes and r has exactly λ_0 bits. The procedure then outputs r if the parsing is successful (and $\perp$ if the parsing fails).

Here, the condition that $\lambda_0 \leq \lambda_1 - 88$ ensures that the random padding P has at least 8 bytes, as required by the standard [33]. We also assume in our description that KEM decapsulation involves performing a strict set of parsing checks.

The assumption that RSA PKCS #1v1.5 is OW-PCA is justified in [32, Theorem 1] via a reduction to an RSA-like assumption, known as "partial-domain RSA with decision oracle". The latter, given in [32, Section 2.3], asserts that the RSA permutation is one-way, even given an oracle that is parameterized by $\lambda_0 < \lambda_1$, takes as input (x_0, y), and reports whether the first λ_0 bits of $y^d \bmod N$ equals x_0 or not. A close examination of the proof of [32, Theorem 1] shows that the theorem holds no matter what set of

parsing checks are carried out during decapsulation (so long as the decapsulation algorithm is correct). This is convenient because, as recent work [7] has shown, there is a good deal of variation in how the required parsing is done in different PKCS #1v1.5 implementations.[10]

In TLS-RSA, λ_0 is fixed to 384, reflecting the fixed size of PMS (at 48 bytes) in the TLS specification, while λ_1 (the bit-size of N) is typically 1024 or 2048 in TLS deployments. Jonsson and Kaliski in [32] discuss at some length why the above assumption is reasonable for typical parameters λ_0, λ_1 used in practice. While seemingly strong, it seems hard to avoid using an assumption of this type given the many known weaknesses in RSA-PKCS#1 v1.5. We are not aware of any further significant work studying this assumption. In particular, in spite of the importance of the widely-deployed PKCS #1v1.5 scheme and its use in TLS, to the best of our knowledge no weaknesses on the assumption have been reported since its introduction in [32] over 10 years ago.

The security of TLS-RSA follows from Theorem 1 and Lemma 6:

Theorem 2. *Under the following assumptions:*

- *RSA PKCS #1v1.5 is OW-PCA;*
- PRF *is a secure pseudorandom function;*
- stE *is a sLHAE encryption scheme,*

the TLS-RSA Protocol is a secure SACCE protocol in the random oracle model.

6 TLS-CCA: Instantiations from IND-CCA

Here, we show that if the underlying KEM $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ is IND-CCA-secure, then the tlskem scheme in Section 4.2 is IND-CCCA-secure. Hence, by Theorem 1, the corresponding TLS scheme TLS-CCA is SACCE-secure. Note that the IND-CCA requirement is stronger than OW-PCA security, but unlike the results in Section 5, we do not rely on random oracles here.

IND-CCA security for KEMs [21]. For a stateful adversary $\mathcal{A}$ and a KEM $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, we define the advantage function

$$\mathbf{Adv}^{\mathrm{ind-cca}}_{\mathsf{kem}}(\mathcal{A}) := \Pr\left[b = b' : \begin{array}{l} (\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda); \\ (\psi^*, K^*) \leftarrow \mathsf{Enc}(\mathrm{PK}); \\ K_0 := K^*; K_1 \leftarrow_{\mathrm{R}} \{0,1\}^\lambda; b \leftarrow_{\mathrm{R}} \{0,1\}; \\ b' \leftarrow \mathcal{A}^{\mathsf{Dec}(\mathrm{SK},\cdot)}(\psi^*, K_b) \end{array} \right] - \frac{1}{2}$$

with the restriction that the adversary only queries the oracle on $\psi \neq \psi^*$. A KEM is said to be *indistinguishable against chosen ciphertext attacks* (IND-CCA) if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathrm{ind-cca}}_{\mathsf{kem}}(\mathcal{A})$ is a negligible function in λ.

6.1 IND-CCCA from IND-CCA

Lemma 7 (IND-CCCA from IND-CCA). *If* $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ *is IND-CCA secure and* PRF *and* Kdf *are pseudorandom functions, then the LKEM* tlskem *is IND-CCCA secure. That is, for any adversary* $\mathcal{A}$

[10] This property of the proof would also allow us to incorporate into our analysis the additional check from the TLS specification that the leading 2 bytes of r should be an encoding of the TLS protocol version as sent by the client, at the cost of reducing λ_0, the bit-size of r, by 16. However, we omit this fine detail.

that makes at most Q decryption queries, there exists adversaries $\mathcal{A}_1, \mathcal{A}_2$ such that

$$\mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{tlskem,tls.Pred}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ind-cca}}_{\mathsf{kem}}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Kdf}}(\mathcal{A}_2) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A}_3) + Q \cdot 2^{-\lambda}$$

Moreover, the running times of $\mathcal{A}_1, \mathcal{A}_2$ are roughly that of $\mathcal{A}$.

We begin with a high-level overview of the proof. Let $(\eta^* \| \textsc{cert}^*_S, \textsc{cres}^*)$ denote the challenge ciphertext and let $(\eta \| \textsc{cert}_S, \textsc{cres}, \textsc{cfin})$ denote a query to the CDec oracle. Recall that the adversary also gets either K_0 or K_1 where K_0 is the real key and K_1 is a random key. We proceed via a case analysis:

- if $\textsc{cres}^* \neq \textsc{cres}$, then we can easily answer the decryption oracle using the decryption oracle for the underlying IND-CCA scheme;
- if $\textsc{cres}^* = \textsc{cres}$, then we want to exploit pseudorandomness of $\mathsf{Kdf}(\textsc{pms}^*, \cdot)$ to argue that the adversary cannot provide a consistent $\textsc{cfin}$, where $\textsc{pms}^* := \mathsf{F}_S(\textsc{sk}, \textsc{cres}^*)$. This step requires some care because $\textsc{cres}^*$ leaks information about the seed $\textsc{pms}^*$.

Proof. We proceed via a series of games. We start with Game 0, where the challenger proceeds as in the real IND-CCCA game (i.e, K_0 is a real key and K_1 is a random key) and end up with a game where both K_0 and K_1 are chosen uniformly at random. We use $\mathbf{Adv}_0, \mathbf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc. Let $\textsc{pms}^*, \textsc{cres}^*, \textsc{ms}^*, \ldots$ denote the values used to compute the challenge ciphertext in the IND-CCCA experiment. We assume that the adversary $\mathcal{A}$ makes at most Q queries to the decryption oracle.

Game 0. Real experiment.

Game 1. Replace $\textsc{pms}^*$ with $\textsc{pms}^* \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$. That is, for $i = 1, 2, \ldots, Q$, we simulate the response to the i'th decryption query $(\eta \| \textsc{cert}_S, \textsc{cres}, \textsc{cfin})$ to CDec as follows:

1. abort if $(\eta \| \textsc{cert}_S, \textsc{cres})$ equals $(\eta^* \| \textsc{cert}^*_S, \textsc{cres}^*)$ as in Game 0;
2. if $\textsc{cres} \neq \textsc{cres}^*$, query the decryption oracle $\mathsf{F}_S(\textsc{sk}, \cdot)$ on input $\textsc{cres}$ to compute $\textsc{pms}$ and proceed as in Game 0.
3. if $\textsc{cres} = \textsc{cres}^*$ and $(\eta, \textsc{cert}_S) \neq (\eta^*, \textsc{cert}^*_S)$, proceed as in Game 0 using $\mathsf{Kdf}(\textsc{pms}^*, \cdot)$.

It is straight-forward to construct an adversary $\mathcal{A}_1$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 + \mathbf{Adv}^{\mathrm{ind-cca}}_{\mathsf{kem}}(\mathcal{A}_1)$$

Game 2. Replace $\textsc{ms}^*$ with $\textsc{ms}^* \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$. It follows immediately from pseudorandomness of $\mathsf{Kdf}(\textsc{pms}^*, \cdot)$, that there exists an adversary $\mathcal{A}_2$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Kdf}}(\mathcal{A}_2)$$

Game 3. We replace $\mathsf{PRF}(\textsc{ms}^*, \cdot)$ with a truly random function $\mathsf{R}(\cdot)$ in the computation of the challenge ciphertext and challenge keys, as well as in responding to all decryption queries satisfying $\textsc{cres} = \textsc{cres}^*$ and $(\eta, \textsc{cert}_S) \neq (\eta^*, \textsc{cert}^*_S)$. It follows immediately from pseudorandomness of $\mathsf{PRF}(\textsc{ms}^*, \cdot)$, that there exists an adversary $\mathcal{A}_3$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_2 \leq \mathbf{Adv}_3 + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A}_3)$$

Game 4. We modify the decryption oracle CDec in the previous game as follows: if $\text{CRES} = \text{CRES}^*$ and $(\eta, \text{CERT}_S) \neq (\eta^*, \text{CERT}_S^*)$, always output $\perp$; otherwise, proceed as in Game 3. We claim that

$$\mathbf{Adv}_3 \leq \mathbf{Adv}_4 + Q \cdot 2^{-\lambda}$$

Whenever $\text{CRES} = \text{CRES}^*$ and $(\eta, \text{CERT}_S) \neq (\eta^*, \text{CERT}_S^*)$, the value $\mathsf{R}(1\|\eta\|\text{CERT}_S\|\text{CRES}^*)$ is statistically random from the view-point of the adversary, so the probability that the adversary provides a ciphertext query to CDec with a valid UCFIN is at most $2^{-\lambda}$.

Observe that in Game 4, the decryption queries reveal no information whatsoever about $\mathsf{R}(\cdot)$. Therefore, the three values

$$(\text{UCFIN}^*, \text{AKEY}^*, \text{USFIN}^*) := (\mathsf{R}(1\|\eta^*\|\text{CERT}_S^*\|\text{CRES}^*), \mathsf{R}(0\|\eta^*), \mathsf{R}(2\|\eta^*\|\text{CERT}_S^*\|\text{CRES}^*\|\text{UCFIN}^*))$$

are independent and uniformly random strings from the view-point of the adversary. That is, $K_0 := \text{AKEY}^*\|\text{USFIN}^*\|\text{UCFIN}^*$ has the uniform distribution, which is the same as that of K_1. Therefore, the view of the adversary is statistically independent of the challenge bit b. Therefore, $\mathbf{Adv}_4 = 0$. The claim follows readily. □

7 TLS-DH: Instantiation from PRF-ODH

We prove the security of TLS-DH following our methodology: We show that the KEM $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ that instantiates this mode in accordance with our generic representation of TLS (Figure 2) induces a labeled KEM, dh.tlskem, that is IND-CCCA secure. Then, by Theorem 1 we conclude that TLS-DH is a secure SACCE protocol (in Section 8 we extend this to the case of mutual authentication). Finally, we apply these results to TLS-DHE, namely, when both client and server provide ephemeral DH keys.

TLS-DH KEM. Let $G = \langle g \rangle$ be a cyclic group of prime order q generated by an element g. We define the TLS-DH KEM $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ via the following three algorithms.

- $\mathsf{KeyGen}(1^\lambda)$: Set $(\text{PK}, \text{SK}) := (g^v, v), v \leftarrow_R \mathbb{Z}_q, |q| = \lambda$.
- $\mathsf{F}_C(\text{PK} = g^v)$: Set $(\psi, K) := (g^u, g^{uv}), u \leftarrow_R \mathbb{Z}_q$.
- $\mathsf{F}_S(\text{SK} = v, h)$: Check that $h \in G$, if yes, output h^v, else output $\perp$ (reject).

7.1 Security of TLS-DH in random oracle model

Before proving the security of TLS-DH in the standard model we observe that in the random oracle model this security readily follows from Lemma 6 under the Strong Diffie-Hellman assumption.

Strong Diffie-Hellman assumption. The Strong DH assumption [1] asserts that computing g^{uv} given (g, g^v, g^u) is hard on average, even given access to a verification oracle $(g, g^v, \cdot, \cdot)$ for DDH tuples. In other words, CDH is hard given a DDH oracle. This immediately implies that the TLS-DH KEM $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$ is OW-PCA. Note that the Strong DH assumption is weaker than (that is, implied by) the Gap (Hashed) DH assumption; moreover, if the hash function is implemented via random oracle, then the two assumptions are equivalent.

Lemma 8. *Under the Strong-DH assumption in the group G, if one models* Kdf *as a random oracle, the TLS-DH KEM is OW-PCA, and therefore* dh.tlskem *(see below) is IND-CCCA and TLS-DH is a secure SACCE protocol in the random oracle model.*

7.2 Security of TLS-DH in the standard model

We show the security of TLS-DH in the standard model based on the PRF-ODH assumption on the function Kdf. The assumption, which we present next, is an adaptation of the Oracle Diffie-Hellman (ODH) assumption [1] to the PRF setting and was introduced in [31] for their proof of TLS-DHE. While [31] argued about the difficulties of proving TLS-DHE without this assumption, we show that at least for TLS-DH (with static server key, ephemeral client key, and no client authentication), the assumption is *provably necessary* by constructing secure pseudorandom functions for which the PRF-ODH assumption does not hold and with which TLS-DH violates ACCE security (see Appendix C).

The PRF-ODH Game. Let $G = \langle g \rangle$ be a cyclic group of order q generated by an element g. Let f be a PRF family that accepts as keys elements from G and outputs strings of length λ. Given $v \in \mathbb{Z}_q$, we define an oracle ODH_v that acts on pairs (g^u, α), where $g^u \in G$ and α is in the domain of f, and outputs $\mathsf{ODH}_v(g^u, \alpha) = f_{g^{uv}}(\alpha)$. Next, we describe a PRF-ODH game between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$.

- Challenger $\mathcal{C}$ chooses (v, g^v) and gives g^v to $\mathcal{A}$.
- $\mathcal{A}$ presents queries to $\mathcal{C}$ of the form (h, α), which $\mathcal{C}$ answers by first checking that $h \in G$ (if not, it aborts) and then returning $\mathsf{ODH}_v(h, \alpha) = f_{h^v}(\alpha)$.
  Note: Since inputs to ODH are checked to be in G we will denote these inputs as g^u even though $\mathcal{A}$ does not have to know u.
- At some point $\mathcal{A}$ chooses a test value $\bar{\alpha}$ to which $\mathcal{C}$ responds with a pair $(g^{\bar{u}}, z_b)$ where $\bar{u} \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $z_0 = f_{g^{\bar{u}v}}(\bar{\alpha})$, $z_1 \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$, $b \leftarrow_{\mathrm{R}} \{0,1\}$.
- Attacker $\mathcal{A}$ keeps querying the ODH oracle via $\mathcal{C}$ except that it is not allowed to query the pair $(g^{\bar{u}}, \bar{\alpha})$.
- At the end of its run, $\mathcal{A}$ outputs a bit b'.

We say that $\mathcal{A}$ wins the PRF-ODH game if $b' = b$, and denote by $\mathbf{Adv}^{\mathrm{odh}}_f(\mathcal{A})$ its advantage.

PRF-ODH Assumption. *We say that the PRF-ODH assumption holds for function f if for all efficient PRF-ODH adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathrm{odh}}_f(\mathcal{A})$ is negligible. We call f PRF-ODH secure.*

Differences with the PRF-ODH assumption from [31]. The assumption introduced in [31] is a specialization of the above assumption to the case of a single query from $\mathcal{A}$ (asked after $\mathcal{C}$ outputs the challenge $(g^{\bar{u}}, z_b)$). The reason for the milder assumption in [31] is that they only apply it to the case of ephemeral DH where the attacker can make a single query to the oracle (this is also the case in our analysis of TLS-DHE). For use with static DH, as in TLS-DH, one needs the general version with a variable number of queries (as in the original ODH assumption from [1]). Another difference with the use in [31] is that they do not allow queries which contain the value $g^{\bar{u}}$ while we do as long as $\alpha \neq \bar{\alpha}$. Finally, in the ephemeral case (single use exponents) there is no need to test group membership.

Lemma 9 (IND-CCCA from PRF-ODH). *If* Kdf *is PRF-ODH secure and* PRF *is a pseudorandom function, then the LKEM* dh.tlskem *is IND-CCCA secure. That is, for any adversary $\mathcal{A}$ that makes at most Q decryption queries, there exists adversaries $\mathcal{A}_1, \mathcal{A}_2$ such that*

$$\mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{odh}}_{\mathrm{Kdf}}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A}_2) + Q \cdot 2^{-\lambda}$$

Moreover, the running times of $\mathcal{A}_1, \mathcal{A}_2$ are roughly that of $\mathcal{A}$.

The proof is similar to that for Lemma 7: roughly speaking, the ODH oracle plays a role similar to the decryption oracle in the IND-CCA security game. The main difference is that we use the ODH oracle to handle CDec queries for $(\eta, \text{CRES}) \neq (\eta^*, \text{CRES}^*)$ whereas we use the IND-CCA decryption oracle to handle queries $\text{CRES} \neq \text{CRES}^*$.

Proof. We proceed via a series of games. We start with Game 0, where the challenger proceeds as in the real IND-CCCA game (i.e, K_0 is a real key and K_1 is a random key) and end up with a game where both K_0 and K_1 are chosen uniformly at random. We use $\mathbf{Adv}_0, \mathbf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc. Let $\eta^*, \text{CERT}^*_S, \text{PMS}^*, \text{CRES}^*, \text{MS}^*, \ldots$ denote the values used to compute the challenge ciphertext in the IND-CCCA experiment. We assume that the adversary $\mathcal{A}$ makes at most Q queries to the decryption oracle.

Game 0. Real experiment.

Game 1. Replace $\text{MS}^* := \mathsf{Kdf}(\text{PMS}^*, \eta^*)$ with $\text{MS}^* \leftarrow_{\text{R}} \{0,1\}^\lambda$. That is, for $i = 1, 2, \ldots, Q$, we simulate the response to the i'th decryption query $(\eta \| \text{CERT}_S, \text{CRES}, \text{CFIN})$ to CDec as follows:

1. abort if $(\eta \| \text{CERT}_S, \text{CRES})$ equals $(\eta^* \| \text{CERT}^*_S, \text{CRES}^*)$ as in Game 0;
2. if $(\eta, \text{CRES}) \neq (\eta^*, \text{CRES}^*)$, query the ODH oracle on input (η, CRES) to compute MS and proceed as in Game 0.
3. if $(\eta, \text{CRES}) = (\eta^*, \text{CRES}^*)$ and $\text{CERT}_S \neq \text{CERT}^*_S$, proceed as in Game 0 using $\mathsf{PRF}(\text{MS}^*, \cdot)$.

It is straight-forward to construct an adversary $\mathcal{A}_1$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 + \mathbf{Adv}^{\text{odh}}_{\text{Kdf}}(\mathcal{A}_1)$$

Game 2. We replace $\mathsf{PRF}(\text{MS}^*, \cdot)$ with a truly random function $\mathsf{R}(\cdot)$ in the computation of the challenge ciphertext and challenge keys, as well as in responding to all decryption queries satisfying $(\eta, \text{CRES}) = (\eta^*, \text{CRES}^*)$ and $\text{CERT}_S \neq \text{CERT}^*_S$. It follows immediately from pseudorandomness, that there exists an adversary $\mathcal{A}_2$ whose running time is roughly that of $\mathcal{A}$, such that

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{prf}}_{\mathsf{PRF}}(\mathcal{A}_2)$$

Game 3. We modify the decryption oracle CDec in the previous game as follows: if $(\eta, \text{CRES}) = (\eta^*, \text{CRES}^*)$ and $\text{CERT}_S \neq \text{CERT}^*_S$, always output $\perp$; otherwise, proceed as in Game 2. We claim that

$$\mathbf{Adv}_2 \leq \mathbf{Adv}_3 + Q \cdot 2^{-\lambda}$$

Whenever $(\eta, \text{CRES}) = (\eta^*, \text{CRES}^*)$ and $\text{CERT}_S \neq \text{CERT}^*_S$, the value $\mathsf{R}(1 \| \eta \| \text{CERT}_S \| \text{CRES}^*)$ is statistically random from the view-point of the adversary, so the probability that the adversary provides a ciphertext query with a valid UCFIN is at most $2^{-\lambda}$.

Observe that in Game 3, the decryption queries reveal no information whatsoever about $\mathsf{R}(\cdot)$. Therefore, the three values

$$(\text{UCFIN}^*, \text{AKEY}^*, \text{USFIN}^*) := (\mathsf{R}(1 \| \eta^* \| \text{CERT}^*_S \| \text{CRES}^*), \mathsf{R}(0 \| \eta^*), \mathsf{R}(2 \| \eta^* \| \text{CERT}^*_S \| \text{CRES}^* \| \text{UCFIN}^*))$$

are independent and uniformly random strings from the view-point of the adversary. That is, $K_0 := \text{AKEY}^* \| \text{USFIN}^* \| \text{UCFIN}^*$ has the uniform distribution, which is the same as that of K_1. Therefore, the view of the adversary is statistically independent of the challenge bit b. Therefore, $\mathbf{Adv}_3 = 0$. The claim follows readily. □

7.3 Application to SACCE security of TLS-DH and TLS-DHE

Combining Lemma 9 with Theorem 1 we obtain the following:

Theorem 3. *Protocol TLS-DH obtained by instatiating the generic TLS protocol from Figure 2 with the above defined TLS-DH KEM is a secure SACCE protocol provided* Kdf *is PRF-ODH,* PRF *is a secure pseudorandom function, and* stE *is an sLHAE-secure encryption scheme.*

Extension to TLS-DHE with server-signed ephemeral DH (and no client authentication). In this variant of TLS-DH, the certified server's public key corresponds to a signature algorithm and the DH value, typically an ephemeral one, is signed by the server itself. All other details are exactly as in TLS-DH. The security of this protocol follows from the analysis of TLS-DH by replacing $\text{CERT}_S = \{\text{ID}_S, \text{PK} = g^v\}_{\text{CA}}$ with $\text{CERT}_S = (\text{CERTSIG}_S, \mathsf{sig}_S(g^v, \ldots), g^v)$, where $\text{CERTSIG}_S = \{\text{ID}_S, \text{PKSIG}_S\}_{\text{CA}}$, PKSIG_S is a public key of S for a signature scheme, and sig_S is a signature produced by S under the corresponding signature key. In other words, instead of a single DH certificate, there is now a two-certificate chain[11]. Note that in the TLS-DHE protocol, the signature by S includes elements other than g^v, in particular the client's nonce which provides freshness to the signature. However, it follows from our analysis that, for SACCE security, the freshness of the signature is not essential (although this property may be desirable for other purposes). We note that [31] provided a specialized proof of TLS-DHE with client authentication. We obtain a proof for that particular case in Section 8 (Corollary 1). However, while [31] show forward security we do not include this (important) property in our general treatment as it is not achieved by any of the other TLS modes.

8 The TLS Handshake Protocol with Mutual Authentication

Here we augment server-only authentication, the SACCE model, with client authentication to obtain mutual authentication. In this setting the client possesses a signature key and uses it to authenticate to the server. We sketch the adaptation of elements from the SACCE treatment to the client-authentication case. We refer to this model as ACCE.

Protocol Changes. Client authentication augments TLS with two messages:

- Client Certificate [RFC 5246, Sec. 7.4.6], which we denote as $\text{CERT}_C = \{\text{ID}_C, \text{PK}_C\}_{\text{CA}}$ where PK_C is a signature (verification) public key; we denote the corresponding signature key by SK_C.
- Certificate Verify [RFC 5246, Sec. 7.4.8], which contains a signature of C on $(\text{CREQ}, \text{SRES}, \text{CERT}_C, \text{CRES})$; we denote it as $\text{CSIG} = \mathsf{sig}(\text{SK}_C; \text{CREQ}, \text{SRES}, \text{CERT}_C, \text{CRES})$.

Thus, the TLS message sequence then becomes $(\text{CREQ}, \text{SRES}, \text{CERT}_C, \text{CRES}, \text{CSIG}, \text{CFIN}, \text{SFIN})$ where $\text{CERT}_C, \text{CRES}, \text{CSIG}, \text{CFIN}$ are all sent without any server message in-between. In addition, both CERT_C and CSIG are added to the prf input for the computation of UCFIN and USFIN. The schematic generic TLS Handshake with mutual authentication is shown in Figure 5.

8.1 ACCE Security Definitions.

We sketch how to modify our SACCE model to obtain a security model suitable for mutually authenticated ACCE protocols. The result is largely the same as the model of [31], except that we do not treat forward security, and [31] does not allow arbitrary registration of corrupted parties.

[11] In this extension of the proof of TLS-DH to TLS-DHE, we use in an essential way the fact that in our model, hence in the proof of TLS-DH, we do not require any validation of the public key by the CA (other than verification of identity). This allows us to accommodate DH keys self-signed by the server (under the CA-certified server's signature key).

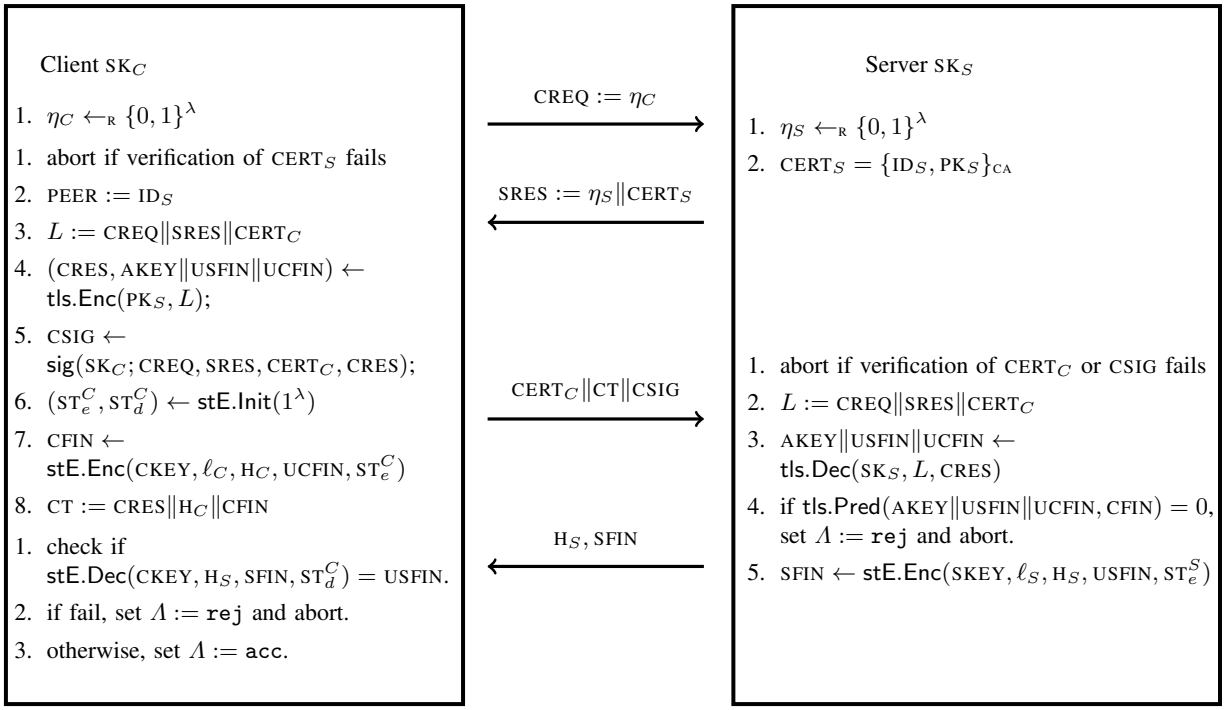

Fig. 5. Basic Generic TLS Handshake Protocol with Client Authentication Parameterized by $(\mathsf{tls.Gen}, \mathsf{tls.Enc}, \mathsf{tls.Dec})$

Model Changes.

- There are multiple clients, each with an identity and a unique public key, and some may be corrupted (though we keep our convention of selective security and a single honest client C).
- Accepted sessions at servers have a defined PEER value.
- We extend the assumption on the certification authority (CA) to client certificates.
- The definition of matching conversations changes only on the required identical transcript prefix: from $(\mathrm{CREQ}, \mathrm{SRES}, \mathrm{CRES})$ to $(\mathrm{CREQ}, \mathrm{SRES}, \mathrm{CERT}_C, \mathrm{CRES})$.
- Correctness definition is augmented with "... and Π_j^S has its PEER variable set to ID_C."
- A Client Authentication requirement is added (this is completely analogous to the server authentication case):
  $\mathbf{Adv}_{\Pi}^{\mathrm{acce-ca}}(\mathcal{A})$ is the probability that when $\mathcal{A}$ terminates, there is a (honest) server S and oracle π_j^S that reaches an accept state with honest $\mathrm{PEER} = \mathrm{ID}_C$ but there is no unique oracle π_i^C for which π_j^S has had a matching conversation with π_i^C.
- The server authentication definition does not change, however, due to the change in matching session definition, the requirement that a client has a matching conversation with a server S means that the matching view of S needs to include CERT_C (or ID_C) in addition to $(\mathrm{CREQ}, \mathrm{SRES}, \mathrm{CRES})$ (this will impact the proof of server authentication).
- The channel security advantage $\mathbf{Adv}_{\Pi}^{\mathrm{acce-ae}}(\mathcal{A})$ is defined exactly as in the SACCE case except that the third condition $P \in \mathcal{C}$ is changed to $P \in \mathcal{C}$ and $P' \in \mathcal{S}$ or $P \in \mathcal{S}$ and $P' \in \mathcal{C}$.

Changes to selective security for client authentication:

Selective server authentication. No change.

Selective client authentication. We require that at the beginning of the security game, the adversary "commits" to (S^*, C^*, j^*) and they must correspond to (S, C, j) in the definition of $\mathbf{Adv}_{\Pi}^{\mathrm{acce-ca}}(\mathcal{A})$.

Selective channel security. We require that at the beginning of the security game for $\mathbf{Adv}^{\mathrm{acce-ae}}_{\Pi}(\mathcal{A})$, the adversary "commits" to (C^*, S^*, i^*, j^*) where either

- (C^*, i^*) correspond to (P, i) in the definition of $\mathbf{Adv}^{\mathrm{acce-ae}}_{\Pi}(\mathcal{A})$ and S^*, j^* are such that $\pi^{C^*}_{i^*}$ has a matching conversation with $\pi^{S^*}_{j^*}$; or
- (S^*, j^*) correspond to (P, i) in the definition of $\mathbf{Adv}^{\mathrm{acce-ae}}_{\Pi}(\mathcal{A})$ and C^*, i^* are such that $\pi^{S^*}_{j^*}$ has a matching conversation with $\pi^{C^*}_{i^*}$.

In selective channel security, the existence of a matching conversation will follow from the proof of server authentication and client authentication, respectively, that will precede the proof of channel security.

We use $\mathbf{Adv}^{\mathrm{s-acce-sa}}_{\Pi}(\mathcal{A}), \mathbf{Adv}^{\mathrm{s-acce-ca}}_{\Pi}(\mathcal{A}), \mathbf{Adv}^{\mathrm{s-acce-ae}}_{\Pi}(\mathcal{A})$ to denote the respective advantage measures in the selective setting.

LKEM Changes. The LKEM from TLS stays exactly the same except that the label $\eta \| \mathrm{CERT}_S$ is replaced with $\eta \| \mathrm{CERT}_S \| \mathrm{CERT}_C$.

8.2 ACCE Security (with mutual authentication)

The following theorem extends Theorem 1 to the mutual authentication setting, namely, when client authentication is added.

Theorem 4. *If* tlskem *is IND-CCCA secure (as defined in Section 4.1),* sig *and* casig *are existentially unforgeable signature schemes, and* stE *is sLHAE-secure (as defined in Appendix B.1), then TLS is ACCE-secure (as defined in Section 8.1).*

Corollary 1. *TLS-RSA, TLS-CCA, TLS-DH and TLS-DHE are all ACCE secure under the assumptions stated in Theorem 2, Lemma 7 and Theorem 3, respectively.*

Server authentication. We begin with server authentication:

Lemma 10 (s-acce-sa). *For any adversary $\mathcal{A}$, there exists adversaries $\mathcal{A}_1, \mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{s-acce-sa}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{sig}}_{\mathsf{casig}}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B}) + (d^2_{max} + 1) \cdot 2^{-\lambda}$$

Moreover, the running times of $\mathcal{A}_1, \mathcal{B}$ are roughly that of $\mathcal{A}$.

Proof (sketch). Similar to the proof of Lemma 2, with changes in the game sequence highlighted in boxes.

Game 0. Real experiment.

Game 1. The challenger proceeds as before, but it aborts if any of the following holds:

- $(\mathrm{ID}^*_S, \mathrm{PK}^*_S) \neq (\mathrm{ID}_S, \mathrm{PK}_S)$; or
- the transcript at any server oracle π^S_j begins with $(\mathrm{CREQ}^*, \mathrm{SRES}^*, \boxed{\mathrm{CERT}^*_C}, \mathrm{CRES}^*)$;
- if the server nonces are not all distinct.

We claim that the first two aborts do not affect the advantage of the adversary (unless the adversary manages to forge the CA's signature on $(\text{ID}^*_S, \text{PK}^*_S)$, an event with probability bounded by $\mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1)$), and the third happens with probability $d^2_{\text{max}} \cdot 2^{-\lambda}$, so

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 + \mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1) + d^2_{\text{max}} \cdot 2^{-\lambda}$$

Game 2. The challenger proceeds as before, but replaces the $(\text{AKEY}, \text{USFIN}, \text{UCFIN})$ computed by $\pi^C_{i^*}$ with a random $(\widetilde{\text{AKEY}}, \widetilde{\text{USFIN}}, \widetilde{\text{UCFIN}}) \leftarrow_{\text{R}} \{0,1\}^{4\lambda}$. That is,

- parse $\widetilde{\text{AKEY}}$ as $\widetilde{\text{CKEY}} \| \widetilde{\text{SKEY}}$;
- $\pi^C_{i^*}$ computes $\text{CT}^* := \text{CRES}^* \| \text{H}^*_C \| \text{CFIN}^*$ at $\pi^C_{i^*}$ by first computing CRES^*, and then using $\widetilde{\text{CKEY}}$ to encrypt $\widetilde{\text{UCFIN}}$.
- $\boxed{\pi^C_{i^*} \text{ computes } \text{CERT}^*_C \text{ and } \text{CSIG}^* \text{ as in the previous game.}}$
- $\pi^C_{i^*}$ decrypts $\text{H}^*_S \| \text{SFIN}^*$ using $\widetilde{\text{SKEY}}$ and checks equality with $\widetilde{\text{USFIN}}$.
- the challenger responds to $\mathsf{Reveal}(\pi^C_{i^*})$ with $\widetilde{\text{AKEY}}$.

By IND-CCCA security of tlskem, we have (analogous to Lemma 3):

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B})$$

Now, we have $\mathbf{Adv}_2 \leq 2^{-\lambda}$, because $\widetilde{\text{USFIN}}$ is truly random from the perspective of $\mathcal{A}$. Putting everything together, we obtain the bound on $\mathbf{Adv}^{\text{s-acce-sa}}_{\Pi}(\mathcal{A})$ as claimed. □

Client authentication. We continue with client authentication:

Lemma 11 (s-acce-ca). *For any adversary $\mathcal{A}$, there exists adversaries $\mathcal{A}_1, \mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{s-acce-ca}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1) + \mathbf{Adv}^{\text{sig}}_{\mathsf{sig}}(\mathcal{B}) + d^2_{max} \cdot 2^{-\lambda}$$

Moreover, the running times of $\mathcal{A}_1, \mathcal{B}$ are roughly that of $\mathcal{A}$.

Proof (sketch). Similar to the proof of Lemma 10. Let j^* denote the target session in the selective ACCE game, and ID_C, PK_C denote the identity and public key of the honest client C. We use $\text{CREQ}^*, \text{SRES}^*, \text{CT}^*, \text{SFIN}^*$ to denote the messages sent or received by $\pi^S_{j^*}$. We proceed via a series of games. We use $\mathbf{Adv}_0, \mathbf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc.

Game 0. Real experiment.

Game 1. The challenger proceeds as before, but it aborts if any of the following holds:

- $(\text{ID}^*_C, \text{PK}^*_C) \neq (\text{ID}_C, \text{PK}_C)$; or
- the transcript at any client oracle π^C_i begins with $(\text{CREQ}^*, \text{SRES}^*, \text{CERT}^*_C, \text{CRES}^*)$;
- if the client nonces are not all distinct.

We claim that the first two aborts do not affect the advantage of the adversary (unless the adversary manages to forge the CA's signature on on $(\text{ID}^*_C, \text{PK}^*_C)$, an event with probability bounded by $\mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1)$), and the third happens with probability $d^2_{\text{max}} \cdot 2^{-\lambda}$, so

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 + \mathbf{Adv}^{\text{sig}}_{\mathsf{casig}}(\mathcal{A}_1) + d^2_{\text{max}} \cdot 2^{-\lambda}$$

To bound $\mathbf{Adv}_1$, we rely on existential unforgeability of sig. That is, we construct $\mathcal{B}$ such that

$$\mathbf{Adv}_1 \leq \mathbf{Adv}^{\mathrm{sig}}_{\mathsf{sig}}(\mathcal{B})$$

$\mathcal{B}$ uses the signing oracle for sig to produce CSIG as a signature for $(\mathrm{CREQ}, \mathrm{SRES}, \mathrm{CERT}_C, \mathrm{CRES}) \neq (\mathrm{CREQ}^*, \mathrm{SRES}^*, \mathrm{CERT}^*_C, \mathrm{CRES}^*)$ at all the client sessions, and where the forgery corresponds to a valid signature on $(\mathrm{CREQ}^*, \mathrm{SRES}^*, \mathrm{CERT}^*_C, \mathrm{CRES}^*)$. Putting everything together, we obtain the bound on $\mathbf{Adv}^{\mathrm{s-acce-ca}}_{\Pi}(\mathcal{A})$ as claimed. □

Channel security. We complete the proof by establishing channel security:

Lemma 12 (s-acce-ae). *For any adversary $\mathcal{A}$, there exists adversaries $\mathcal{B}, \mathcal{A}_2$ and $\mathcal{A}_3$ such that*

$$\mathbf{Adv}^{\mathrm{s-acce-ae}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{lh\text{-}st\text{-}ae}}_{\mathsf{stE}}(\mathcal{A}_2) + \mathbf{Adv}^{\mathrm{lh\text{-}st\text{-}ae}}_{\mathsf{stE}}(\mathcal{A}_3) + d_{max} \cdot 2^{-\lambda}$$

where d_{max} is an upper bound on the number of sessions invoked by $\mathcal{A}$ at any party. Moreover, the running times of $\mathcal{B}, cA_2$ and $\mathcal{A}_3$ are roughly that of $\mathcal{A}$.

Proof (sketch). Similar to the proof of Lemma 4, with changes in the game sequence highlighted in boxes.

Game 0. Real experiment with challenge bit $b = 0$. Henceforth, we neglect the cases where b' is $\perp$.

Game 1. The challenger in this game proceeds as before, but it aborts if any of the following holds:

- any server nonce for $j \neq j^*$ matches that for $\pi^S_{j^*}$; or
- the transcript at server oracle $\pi^S_{j^*}$ does not begin with $(\mathrm{CREQ}^*, \mathrm{SRES}^*, \boxed{\mathrm{CERT}^*_C}, \mathrm{CRES}^*)$.

Thus,

$$\mathbf{Adv}_0 \leq \mathbf{Adv}_1 - d_{\max} \cdot 2^{-\lambda}$$

Game 2. The challenger proceeds as before, except it replaces $(\mathrm{AKEY}, \mathrm{USFIN}, \mathrm{UCFIN})$ computed by $\pi^C_{i^*}$ with a random $(\widetilde{\mathrm{AKEY}}, \widetilde{\mathrm{USFIN}}, \widetilde{\mathrm{UCFIN}}) \leftarrow_{\mathrm{R}} \{0,1\}^{4\lambda}$. Specifically:

- parse $\widetilde{\mathrm{AKEY}}$ as $\widetilde{\mathrm{CKEY}} \| \widetilde{\mathrm{SKEY}}$;
- $\pi^C_{i^*}$ computes CT^* as follows:
  1. generate CRES^* using tls.Enc as in Game 1;
  2. compute $\mathrm{H}^*_C \| \mathrm{CFIN}^*$ by encrypting $\widetilde{\mathrm{UCFIN}}$ with $\widetilde{\mathrm{CKEY}}$;
  3. set $\mathrm{CT}^* := \mathrm{CRES}^* \| \mathrm{H}^*_C \| \mathrm{CFIN}^*$ as in Game 1.
  4. $\boxed{\text{compute } \mathrm{CERT}^*_C \text{ and } \mathrm{CSIG}^* \text{ as in the previous game.}}$
- $\pi^S_{j^*}$ computes the server `Finished` message on input $\mathrm{CT} := \mathrm{CRES} \| \mathrm{H}_C \| \mathrm{CFIN}$ as follows:
  1. if $\mathrm{CT} = \mathrm{CT}^*$: sets $\Lambda :=$ `acc` and responds with the encryption of $\widetilde{\mathrm{USFIN}}$ under $\widetilde{\mathrm{SKEY}}$;
  2. if $\mathrm{CRES} = \mathrm{CRES}^*$ but $\mathrm{H}_C \| \mathrm{CFIN} \neq \mathrm{H}^*_C \| \mathrm{CFIN}^*$: if decrypting $\mathrm{H}_C \| \mathrm{CFIN}$ with $\widetilde{\mathrm{CKEY}}$ yields $\widetilde{\mathrm{UCFIN}}$, sets $\Lambda :=$ `acc` and responds with the encryption of $\widetilde{\mathrm{USFIN}}$ under $\widetilde{\mathrm{SKEY}}$, and otherwise, sets $\Gamma :=$ `rej` and aborts;
  3. if $\mathrm{CRES} \neq \mathrm{CRES}^*$: aborts as in Game 1.
- $\pi^C_{i^*}$ verifies $\mathrm{H}^*_S \| \mathrm{SFIN}^*$ it receives from $\mathcal{A}$ by decrypting with $\widetilde{\mathrm{SKEY}}$ and checking equality with $\widetilde{\mathrm{USFIN}}$, and aborts if verification fails.

– answer $\mathsf{Encrypt}(\pi_{i^*}^C, \ldots)$ and $\mathsf{Decrypt}(\pi_{i^*}^C, \ldots)$ queries using $\widetilde{\text{CKEY}}$.
– answer $\mathsf{Encrypt}(\pi_{j^*}^S, \ldots)$ and $\mathsf{Decrypt}(\pi_{j^*}^S, \ldots)$ queries using $\widetilde{\text{SKEY}}$.

By IND-CCCA security of tlskem, we have (analogous to Lemma 5):

$$\mathbf{Adv}_1 \leq \mathbf{Adv}_2 + \mathbf{Adv}^{\text{ind-ccca}}_{\mathsf{tlskem},\mathsf{tls.Pred}}(\mathcal{B})$$

Game 3. The challenger proceeds as before, except we modify $\pi_{j^*}^S$ as follows:

– $\pi_{j^*}^S$ computes the server `Finished` message on input $\text{CT} := \text{CRES} \| \text{H}_C \| \text{CFIN}$ as follows:

1. if $\text{CT} = \text{CT}^*$, sets $\Lambda := \texttt{acc}$ and responds with the encryption of $\widetilde{\text{USFIN}}$ under $\widetilde{\text{SKEY}}$ as in Game 2;
2. if $\text{CRES} = \text{CRES}^*$ but $\text{H}_C \| \text{CFIN} \neq \text{H}_C^* \| \text{CFIN}^*$, sets $\Gamma := \texttt{rej}$ and abort (this is the only difference from Game 2);
3. if $\text{CRES} \neq \text{CRES}^*$, then abort as in Game 2.

It is straight-forward to construct $\mathcal{A}_2$ so that

$$\mathbf{Adv}_2 \leq \mathbf{Adv}_3 + \mathbf{Adv}^{\text{lh-st-ae}}_{\mathsf{stE}}(\mathcal{A}_2)$$

Putting everything together, we obtain the bound on $\mathbf{Adv}^{\text{s-acce-ae}}_{\Pi}(\mathcal{A})$ as claimed. □

9 Conclusions and Discussion

Establishing the security of a central protocol like TLS is clearly a significant result. The fact that we achieve this in a systematic way that covers the different TLS modes (TLS-RSA, TLS-DH, and TLS-DHE, as well as the hypothetical TLS-CCA), has clear methodological advantages and may be seen as indicating that the core design of the protocol is sound. Yet, we find it important to stress the many shortcomings of the TLS protocol that would be best avoided in future secure channel protocol designs.

On the TLS design. A main weakness in the overall TLS design is the unfortunate interaction of the TLS Handshake and the Record protocols with respect to the `Finished` messages. The interaction arises because of protocol layering: the Handshake Protocol runs as a layer on top of the Record Protocol, but when the `ChangeCipherSpec` message is sent in the middle of the Handshake Protocol, this signals to the Record Protocol that it should start to use the negotiated Record Protocol encryption. The consequence is that the `Finished` messages belonging to the TLS Handshake end up being encrypted at the Record layer. Instead, these two conceptually and functionally different parts of a secure channel protocol can and should be designed as separate components. This separation makes engineering sense for implementing and maintaining the protocol, especially in evolving application settings as is the case with TLS. And it is even more important when it comes to the problem of cryptographic design and analysis of complex protocols like TLS. Indeed, a suitable separation of authenticated key exchange and subsequent use of the exchanged keys to protect data in the secure channel allows greater modularisation of the security analysis, which in turn increases the likelihood that this analysis will be correct and applicable to the actual protocol under study. More specifically, this separation would enable the designer to focus on achieving key indistinguishability for the key exchange component and to take advantage of well-established formal models for key exchange, whilst analysing the subsequent use of the exchanged keys under the assumption that they are truly random (which is the usual assumption when analysing symmetric key protocols). This separation also allows cleaner analysis of further mechanisms such as

the derivation of additional keys for future sessions (as in TLS's session resumption feature) and would makes design errors in protocol extensions, such as the famous TLS renegotiation bug [45, 27], easier to spot.

In addition to the above structural weakness of the protocol, TLS has suffered from the early implementation of its public key encryption mode, TLS-RSA, with RSA PKCS#1v1.5. This led to Bleichenbacher attack on the original SSL protocol that exploited the error message output by the server when receiving an invalid ciphertext. Rather than moving to a CCA-secure implementation of the encryption function (e.g., via RSA-OAEP), the TLS community responded to the attack by keeping RSA PKCS#1v1.5 as the default implementation but disabling the error message in case of a decryption error. Instead, there is now a single error message for both the case of decryption errors and failed authentication (to mask timing information, a server receiving an invalid ciphertext continues the protocol by choosing a random pre-master key). Before the present paper it was unknown whether the hiding of ciphertext invalidity was sufficient to make the protocol secure; we resolve this question positively. Yet, we would like to stress that our proof of security relies crucially on there being no side channel that would reveal the existence of decryption failures to the attacker. While the TLS specification takes care to avoid some explicit forms of leaking this information (as explained above), implementations may still leak a decryption failure (e.g., a secure token may handle RSA PKCS#1v1.5 decryption errors in a way that does leak timing information). This makes the security of the protocol non-robust and shows the clear advantage of using a CCA-secure scheme in TLS that, as we show, would avoid these complications and potential weaknesses. In addition, the fact that the security proof for TLS-CCA does not require a random oracle is a further indication of the robustness that this choice would bring (this is so even in cases where the CCA encryption in use does assume a random oracle, as in the RSA-OAEP case).

The fragility of the TLS-RSA design is further illustrated by the somewhat "accidental" nature of its security. Indeed, the security of this mode relies crucially on the fact that the client's `Finished` message, CFIN, is sent by the client immediately after sending the CRES message (containing the RSA ciphertext in TLS-RSA) and before the server responds with its own `Finished` message SFIN. Had CFIN been omitted or sent after SFIN, TLS-RSA would be completely insecure, e.g. subject to Bleichenbacher's attack. Moreover, omitting CFIN or sending it after receiving SFIN does not change the proof of security of any of the other TLS modes. Actually, setting aside TLS-RSA, it would have been advisable to send CFIN after SFIN as a form of key confirmation by the client once it verified the incoming SFIN. Incidentally, these subtleties about the changes in protocol security that may arise depending on the placement of the `Finished` messages serve as examples of the care one needs to have when analyzing "variants" of TLS. Such variants may say very little about the security of the actual protocol.

An additional weaknesses in the design of TLS is the fact that the master secret is not only used for authentication of the TLS handshake while creating a TLS session but is also stored for authentication and key derivation for resumed sessions. The correct design is to generate three types of keys during the Handshake: a key for authenticating the Handshake itself, application keys for protecting the Record Protocol (data), and (if so desired) a third key that can be used for derivation of further session keys without having to run a full handshake with the cost of public key operations that this incurs. These three keys can be derived using a PRF in a way that the compromise of any of them does not compromise the others. This is the approach taken, for example, in IPsec's IKE protocol [28, 17].

On TLS-RSA vs TLS-DH. A crucial distinction between TLS-RSA and TLS-DH is that publicly verifying validity of CRES is easy in TLS-DH (assuming efficient checking of group membership) but hard in TLS-RSA (where CRES is a RSA PKCS #1v1.5 ciphertext). This distinction underlies Bleichenbacher's attack and explains why it is necessary to treat CRES$\|$CFIN as a monolithic message

in TLS-RSA but not in TLS-DH. Indeed, in applications where the decryption outcome may be leaked (e.g. via side channels), we recommend using TLS-DH instead of TLS-RSA. Even better, TLS-DHE also offers forward security, a feature not offered by either TLS-RSA or TLS-DH.

On our attack model. We caution that, while our work shows that accurate descriptions of already-deployed, complex protocols can be analysed using the provable security paradigm, we only analyse the "cryptographic core" of TLS. This means that our analysis rules out many (but not all) attacks. More specifically:

- Several recent attacks on the TLS Record Protocol are possible because TLS supports symmetric encryption algorithms that have turned out *not* to be sLHAE-secure: see, for example, the BEAST attack [23], the short MAC attack [42], Lucky 13 [3], and the RC4 attacks in [4]. Such attacks can be mounted by the adversary in our security model but are ruled out by our theorem statements, which assume the use of an sLHAE-secure encryption scheme. We note that sLHAE-security *is* achieved when TLS's CBC-mode option is carefully implemented [35, 42], or if the weak RC4 stream cipher is replaced by one whose outputs are indistinguishable from random [35].
- Our description of TLS-RSA includes the standard countermeasures to Bleichenbacher's attack, and our security proof then gives assurance that these countermeasures are effective in the context of the entire TLS protocol. Our work is the first to provide such a guarantee.

On the other hand, we do not treat ciphersuite (re)negotiation nor the TLS Record Protocol's compression or fragmentation features, meaning that, for example, none of the ciphersuite downgrade attack of [36], the renegotiation-based attack of [45], or the CRIME attack [24] are covered by our analysis.

It is certainly conceivable that features omitted from our model, or an interaction between such features and the core of TLS, could lead to new attacks. A list of omitted features can be found in Appendix A. On the other hand, we also believe that our modular approach can be helpful in extending our analysis to encompass more of these features.

A final thought. We believe that one cannot overstate the importance of adopting protocols in practice that have been first rigorously analyzed and proven in a plausible cryptographic model. Such proofs are necessarily limited by the expressiveness of the underlying model and do not guarantee security in every imaginable deployment setting, yet they can serve as a major source of confidence in the soundness of the design. This is particularly important given the practical difficulty in changing protocols when weaknesses are found – and TLS serves as a good example for the latter.

References

[1] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *CT-RSA*, pages 143–158, 2001.

[2] N. AlFardan and K. G. Paterson. Plaintext-recovery attacks against Datagram TLS. In *Network and Distributed System Security Symposium (NDSS 2012)*, 2012.

[3] N. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy*, 2013. URL `www.isg.rhul.ac.uk/tls/Lucky13.html`.

[4] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt. On the security of RC4 in TLS and WPA. In *USENIX Security Symposium*, 2013. URL `www.isg.rhul.ac.uk/tls`.

[5] G. V. Bard. The vulnerability of SSL to chosen plaintext attack. *IACR Cryptology ePrint Archive*, 2004:111, 2004.

[6] G. V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. In *SECRYPT*, pages 99–109, 2006.

[7] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay. Efficient padding oracle attacks on cryptographic hardware. In *CRYPTO*, pages 608–625, 2012.
[8] M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, pages 491–506, 2003.
[9] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.
[10] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In *ACM Conference on Computer and Communications Security*, pages 1–11, 2002.
[11] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Verified cryptographic implementations for TLS. *ACM Trans. Inf. Syst. Secur.*, 15(1):3, 2012.
[12] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with verified cryptographic security. In *IEEE Symposium on Security and Privacy*, 2013. URL http://mitls.rocq.inria.fr/.
[13] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS), May 2006. URL http://www.rfc-editor.org/rfc/rfc4492.txt.
[14] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *CRYPTO*, pages 1–12, 1998.
[15] A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. On symmetric encryption with distinguishable decryption failures. In *FSE*, 2013.
[16] C. Brzuska, M. Fischlin, N. Smart, B. Warinschi, and S. Williams. Less is more: Relaxed yet composable security notions for key exchange. Cryptology ePrint Archive, Report 2012/242, 2012.
[17] E. C. Kaufman. Internet Key Exchange (IKEv2), Dec. 2005. URL http://www.rfc-editor.org/rfc/rfc4306.txt.
[18] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001. See also Cryptology ePrint Archive, Report 2001/040.
[19] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, pages 337–351, 2002. See also Cryptology ePrint Archive, Report 2002/059.
[20] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *CRYPTO*, pages 583–599, 2003.
[21] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002. Also, Cryptology ePrint Archive, Report 2001/085.
[22] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, Aug. 2008. URL http://www.rfc-editor.org/rfc/rfc5246.txt.
[23] T. Duong and J. Rizzo. Here come the ⊕ Ninjas. Unpublished manuscript, 2011.
[24] T. Duong and J. Rizzo. The CRIME attack. Presentation at ekoparty Security Conference, 2012. URL http://www.ekoparty.org/eng/2012/juliano-rizzo.php.
[25] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben. Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In *ACM CCS*, pages 50–61, 2012.
[26] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *ACM CCS*, pages 38–49, 2012.
[27] F. Giesen, F. Kohlar, and D. Stebila. On the security of TLS renegotiation. Cryptology ePrint Archive, Report 2012/630, 2012. http://eprint.iacr.org/.
[28] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), Nov. 1998. URL http://www.rfc-editor.org/rfc/rfc2409.txt.
[29] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *ACM CCS*, pages 2–15, 2005.
[30] D. Hofheinz and E. Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*, pages 553–571, 2007.
[31] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO*, pages 273–293, 2012.
[32] J. Jonsson and B. S. Kaliski Jr. On the security of RSA encryption in TLS. In *CRYPTO*, pages 127–142, 2002.
[33] B. Kaliski. PKCS#1: RSA Encryption Version 1.5, Mar. 1998. URL http://www.rfc-editor.org/rfc/rfc2313.txt.
[34] V. Klíma, O. Pokorný, and T. Rosa. Attacking RSA-based sessions in SSL/TLS. In *CHES*, pages 426–440, 2003.
[35] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *CRYPTO*, pages 310–331, 2001.
[36] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In *ACM CCS*, pages 62–72, 2012.
[37] N. Modadugu and E. Rescorla. The Design and Implementation of Datagram TLS. In *NDSS*. The Internet Society, 2004. ISBN 1-891562-18-5, 1-891562-17-7.
[38] B. Moeller. Security of CBC ciphersuites in SSL/TLS: Problems and countermeasures. Unpublished manuscript, May 2004. http://www.openssl.org/~bodo/tls-cbc.txt.

[39] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In *ASIACRYPT*, pages 55–73, 2008.
[40] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.
[41] T. Okamoto and D. Pointcheval. REACT: Rapid enhanced-security asymmetric cryptosystem transform. In *CT-RSA*, pages 159–175, 2001.
[42] K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *ASIACRYPT*, pages 372–389, 2011.
[43] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.
[44] E. Rescorla and N. Modadugu. Datagram Transport Layer Security, Apr. 2006. URL http://www.rfc-editor.org/rfc/rfc4347.txt.
[45] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), Feb. 2010. URL http://www.ietf.org/rfc/rfc5746.txt.
[46] V. Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. http://eprint.iacr.org/.
[47] S. Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In *EUROCRYPT*, pages 534–546, 2002.
[48] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.

A The TLS Handshake Protocol without Client Authentication

Here, we justify our model for the TLS Handshake Protocol given in Section 2. We focus mostly on the case of TLS-RSA, with modifications for DH modes being sketched. This description is based on the current TLS specification [22].

Client Request. Here, the client sends a request message CREQ containing a value η_C. In the TLS specification, this 32-byte field is formed from a random component and a time-based component; in our analysis, we model it as a random λ-bit string. Also in the specification, as part of the first message (called the Client Hello), the client sends its highest supported TLS protocol version number and a list of ciphersuites indicating which algorithms and key exchange methods it is prepared to use. We omit these features in our model.

Server Response. Here, the server responds with SRES, containing a random λ-bit string η_S and the server's public key PK_S where $(\mathrm{PK}_S, \mathrm{SK}_S) \leftarrow \mathsf{KeyGen}(1^\lambda)$. In the TLS specification, η_S is again a 32-byte field is formed from a random component and a time-based component. Also in the specification, as part of the message containing η_S (called the Server Hello), the server sends a protocol version number and a single ciphersuite from the list offered by the client in the first message. Again, we omit these features from our model. In the TLS specification, the server's public key is optionally accompanied by a certificate chain. We model this via the inclusion of CERT_S which binds the server identity ID_S and public key pk_S.

Client Response. For TLS-RSA, the message CRES carries the client's choice of pre-master secret (PMS) encrypted under the server's public key PK using $\mathsf{F}_C(\mathrm{PK})$. In the TLS specification, PMS is then a 48-byte value whose leading 2 bytes are an encoding of the TLS protocol version number sent previously by the client and whose remaining 46 bytes are random. In our model, we take PMS to be a λ-bit string. For DH ciphersuites, CRES carries the client's choice of DH public value (equivalently, an Elgamal-type KEM key encapsulation).

Client-side Finished and Key Derivation. From PMS, the client derives the master secret $\mathrm{MS} := \mathsf{Kdf}(\mathrm{PMS}, \eta_C \| \eta_S)$ via an application of pseudo-random function Kdf, and then deletes PMS. From the TLS 1.2 specification [22]:

```
master_secret = PRF(pre_master_secret, ‘‘master secret’’,
                    ClientHello.random + ServerHello.random)[0..47];
```

This step is needed in practice to allow for different forms of PMS coming from different key establishment methods supported by TLS. In our analysis of TLS-RSA, we model the function Kdf by a random oracle, and for all other modes that we analyze it as a PRF. The client then computes the client `Finished` message UCFIN by applying the PRF to a hash of the previous hanshake messages (in our notation, these are the messages CREQ, SRES, CRES). From [22]:

```
PRF(master_secret, finished_label, Hash(handshake_messages))
```

It also computes the application key $\text{AKEY} = \text{CKEY}\|\text{SKEY}$ via further pseudo-random function applications.

```
key_block = PRF(SecurityParameters.master_secret, ‘‘key expansion’’,
  SecurityParameters.server_random + SecurityParameters.client_random);
```

Each of CKEY and SKEY is then parsed to provide a MAC key, an encryption key, and an IV (if needed), to be used in TLS's stateful authenticated encryption scheme stE.

Client Finished Message. The client then sends the client `Finished` message CFIN which consists of a header field H_C followed by the symmetric encryption of UCFIN under stE using the key CKEY, header H_C, and target ciphertext length ℓ_C. In the TLS specification, the header field has a specific, 5-byte format, consisting of a 1-byte content type field, a 2-byte protocol version field and a 2-byte length field. Because the content is a Handshake Protocol message, the content-type field has value 0x16. The target ciphertext length ℓ_C reflects that the client may use the Record Protocol's variable length padding feature. In the TLS specification, the client sends a change cipher spec message before this client `Finished` message to indicate to the server that it is switching to use the newly established application keys in the Record Protocol. We omit this message from our analysis only for ease of presentation. We stress that in our model, we allow the honest client the same flexibility in choosing H_C and ℓ_C; in particular, H_C and ℓ_C are not fixed by the transcript of the protocol up to CRES. For simplicity, we also assume that the server does not validate H_C (checking, for example, that the content-type field has value 0x16) but just interprets the received message CFIN as containing a Handshake Protocol message according to its internal state at the point of receipt.

Server-side Processing. For TLS-RSA, when the server receives CRES, it decrypts it using its private key SK in an attempt to recover PMS. In earlier versions of the protocol (namely SSL 3.0), an error message was returned immediately if the RSA decryption failed. This enabled Bleichenbacher's attack [14], which exploits the fact that RSA PKCS#1v1.5 is *not* CCA-secure to recover the PMS value chosen by an honest client in a target protocol session. In TLS 1.0 and higher, the specification mandates suppressing any decryption error message, setting PMS to a random value, and then carrying on with the Handshake if decryption of CRES fails. We model this, with the server setting PMS to be a random λ-bit string if decryption fails, and then proceeding to process the CFIN message when it arrives, using the derived master secret MS. This processing involves use of stE's decryption algorithm, followed by recomputing the value UCFIN and comparing it to the received value. For simplicity, we keep the operation of replacing PMS by a random value upon an invalid ciphertext in our generic description of TLS, yet we note that this operation is not necessary for the security of the DH modes or TLS-CCA. For DH ciphersuites, the server computes $\text{PMS} \leftarrow \mathsf{F}_S(\text{SK}_S, \text{CRES})$, that is, it completes the DH key exchange based on the received DH value and its static private key SK_S to compute PMS. It then proceeds to process the CFIN message as above.

Server Finished Message. If these steps succeed, then the server prepares and sends the SFIN message, which consists of a header field H_S followed by the symmetric encryption of USFIN under stE using

the key SKEY. Similar considerations apply to H_S and ℓ_S as to H_C and ℓ_C. In the TLS specification, the server first sends a change cipher spec message; we omit this message from our analysis. The server `Finished` message is received at the client and processed in the same way as the client `Finished` message is processed at the server.

Pseudorandom Function. According to [22], both the key derivation function Kdf and the pseudorandom function PRF are derived from a single pseudorandom function HMAC.PRF based on HMAC with different prefixes, namely:

$$\begin{aligned}
\mathsf{Kdf}(\mathrm{PMS}, x) &:= \mathsf{HMAC.PRF}(\mathrm{PMS}, \texttt{master secret}\|x)\\
\mathsf{PRF}(\mathrm{MS}, 0\|x) &:= \mathsf{HMAC.PRF}(\mathrm{MS}, \texttt{key expansion}\|x)\\
\mathsf{PRF}(\mathrm{MS}, 1\|x) &:= \mathsf{HMAC.PRF}(\mathrm{MS}, \texttt{client finish}\|\mathsf{H}(x))\\
\mathsf{PRF}(\mathrm{MS}, 2\|x) &:= \mathsf{HMAC.PRF}(\mathrm{MS}, \texttt{server finish}\|\mathsf{H}(x))
\end{aligned}$$

Here, $\mathsf{H}(\cdot)$ denotes a collision-resistant hash function.

Important note. The use of a collision-resistant hash function H inside the specification of the function PRF means that the quantified PRF security of PRF assumed across the paper (represented by $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}$) already includes the security factor coming from the (assumed) collision resistance of H, hence there is no explicit quantification of collision resistance in our formal claims.

TLS Record Protocol. Assuming the TLS Handshake protocol concludes successfully at both client and server, the two parties then continue to use the application key $\mathrm{AKEY} = \mathrm{CKEY}\|\mathrm{SKEY}$ in the TLS Record Protocol. Specifically, the client uses CKEY to encrypt messages to the server, and the server uses the same key CKEY to decrypt these messages. Similarly, the server uses SKEY to encrypt messages to the client, and the client uses the same key SKEY to decrypt these messages. As noted above, the client and server `Finished` messages are already protected in this way. As in [31], we model the TLS Record Protocol via a stateful AEAD scheme stE which we will assume to be sLHAE-secure. The headers H used in this phase are all 5 bytes in length, as for the `Finished` messages, but the content type field is set to 0x17 instead of 0x16. In the TLS specification, the client and server encryption and decryption states $\mathrm{ST}^C_e, \mathrm{ST}^C_d, \mathrm{ST}^S_e, \mathrm{ST}^S_d$ are 64-bit counters whose values are zero for the `Finished` messages and which are incremented for every subsequent Record Protocol message sent or received until the maximum value is reached, at which point encryption and decryption algorithms are assumed to output $\perp$ (in-line with the TLS specification which forces rekeying at this point).

Omissions and Simplifications In order to focus on the cryptographic core of TLS, we omit several features of the TLS Handshake Protocol, including the `ChangeCipherSpec` messages, and the negotiation of protocol versions and ciphersuites. The `ChangeCipherSpec` messages are trivially added to our description.

The omission of negotiation means that we do not model ciphersuite downgrade attacks (c.f. [48, 36]) or protocol version rollback attacks. There is little formal analysis of these classes of attacks in the literature, and we do no worse than previous authors analysing TLS [39, 31] in this regard. This does however mean that our treatment of the different key establishment methods provides only "stand-alone" security guarantees for these different methods. On the other hand, if we were to add ciphersuite negotiation to our our treatment in the obvious way, i.e., by adding the offering and choice of ciphertexts to the CREQ and SRES messages, respectively, then our approach (in particular our definition of matching conversations and its use in our security definitions) would at least guarantee that *if the TLS mode agreed*

upon by the parties is secure then those parties would gain an assurance as to their agreement on the Handshake messages and the validity of negotiation.

We model neither session resumption nor ciphersuite renegotiation, the latter also being the subject of recent attacks and fixes [45] and formal analysis [27]. We also omit consideration of TLS extensions. Since we do not treat protocol version numbers, we also omit any special treatment of the leading 2 bytes of PMS. The TLS specification [22] indicates that care is needed in their processing in order to avoid certain extensions of the Bleichenbacher attack [34].

Our choice of modelling the TLS Record Protocol as a stateful symmetric encryption scheme means that we do not treat TLS's fragmentation and compression features, with the latter recently being exploited in attacks [24]. We also simplify TLS's treatment of error messages and channel management, omitting detailed consideration of the TLS Alert protocol which handles such issues. (More generally, we do not model the fact that the TLS Record Protocol actually supports multiple upper-layer protocols, as identified by the content-type field in its message headers, but just treat all of its messages uniformly.)

Finally, we treat all keys and random numbers as being λ bits in length, where λ is our security parameter. This modelling choice is easily upgraded to allow for different lengths for the various fields.

B Preliminary Definitions

B.1 Stateful Length-Hiding Authenticated Encryption

We follow [42] in upgrading the stateful CCA security notion of Bellare, Kohno, and Namprempre [10] to obtain the stateful length-hiding authenticated encryption (sLHAE) security notion.

A *stateful (symmetric) AEAD scheme* $\mathsf{stE} = (\mathsf{stE.Gen}, \mathsf{stE.Init}, \mathsf{stE.Enc}, \mathsf{stE.Dec})$ is a quadruple of algorithms. The probabilistic algorithm $\mathsf{stE.Gen}$ samples a key K from a finite and non-empty set $\mathcal{K}$. The deterministic algorithm $\mathsf{stE.Init}$ initializes two states ST_e, ST_d, one for encryption and one for decryption. The *encryption algorithm* $\mathsf{stE.Enc}$ takes an input $(K, \ell, \mathrm{H}, m, \mathrm{ST}_e) \in \mathcal{K} \times \mathbb{N} \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ (the key, output length, associated data, message, and encryption state) and outputs $(c, \mathrm{ST}_e{}') \in (\{0,1\}^* \cup \perp) \times \{0,1\}^*$ such that $|c| = \ell$ if $c \neq \perp$. Here, then, ℓ denotes the desired ciphertext length, while $\mathrm{ST}_e{}'$ denotes an updated encryption state. The *decryption algorithm* $\mathsf{stE.Dec}$ takes an input $(K, \mathrm{H}, c, \mathrm{ST}_d) \in \mathcal{K} \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ (the key, associated data, ciphertext, and decryption state) and outputs $(m, \mathrm{ST}_d{}') \in (\{0,1\}^* \cup \perp) \times \{0,1\}^*$. The encryption algorithm can be probabilistic while decryption is always deterministic. We assume that there are sets $\mathcal{H} \subseteq \{0,1\}^*$ (the header space), $\mathcal{L} \subseteq \mathbb{N}$ (the requested length space), $\mathcal{M} \subseteq \{0,1\}^*$ (the message space), $\mathcal{ST} \subseteq \{0,1\}^*$ (the state space) such that for all $K \in \mathcal{K}$ and $\mathrm{ST}_e \in \mathcal{ST}$ it holds that $\Pr[\mathsf{stE.Enc}(K, \ell, \mathrm{H}, m, \mathrm{ST}_e) \in \{0,1\}^*] = 1$ if $(\ell, \mathrm{H}, m, \mathrm{ST}_e) \in \mathcal{L} \times \mathcal{H} \times \mathcal{M} \times \mathcal{ST}$ and $\Pr[\mathsf{stE.Enc}(K, \ell, \mathrm{H}, m, \mathrm{ST}_e) = \perp] = 1$ if $(\ell, \mathrm{H}, m, \mathrm{ST}_e) \notin \mathcal{L} \times \mathcal{H} \times \mathcal{M} \times \mathcal{ST}$. For correctness, we require that, if a key is produced by running $\mathsf{stE.Gen}$ and initial states ${\mathrm{ST}_e}^0$,${\mathrm{ST}_d}^0$ are output by $\mathsf{stE.Init}$, and the sequence of encryptions $(c_i, {\mathrm{ST}_e}^{i+1}) \leftarrow \mathsf{stE.Enc}(K, \ell_i, \mathrm{H}_i, m_i, {\mathrm{ST}_e}^i)$ is such that no c_i is equal to $\perp$, then the sequence of decryptions $(m'_i, {\mathrm{ST}_d}^{i+1}) \leftarrow \mathsf{stE.Dec}(K, \mathrm{H}_i, c_i, {\mathrm{ST}_d}^i)$ is such that $m'_i = m_i$ for each $i \geq 0$.

We further make a restriction that whether or not $\mathsf{stE.Enc}$ returns $\perp$ does not depend on the message in any way other than its length (all other inputs kept equal). Formally, for all keys $(K, \ell, \mathrm{H}, \mathrm{ST}_e) \in \mathcal{K} \times \mathcal{L} \times \mathcal{H} \times \mathcal{ST}$ and for all $m, m' \in \mathcal{M}$ such that $|m| = |m'|$ it holds that for all coins $\mathsf{stE.Enc}(K, \ell, \mathrm{H}, m, \mathrm{ST}_e) = \perp$ iff $\mathsf{stE.Enc}(K, \ell, \mathrm{H}, m', \mathrm{ST}_e) = \perp$.

We recall from [42] that this syntax supports encryption schemes that return variable-length ciphertexts of the same plaintext m, with ℓ denoting the desired ciphertext length. Notice that if ℓ and m are such that encryption would return $\perp$ (e.g. because $\ell < |m|$, or the encryption algorithm does not support ciphertexts of length ℓ), then it does so always.

main $\text{sLHAE}_{\mathsf{stE}}$:
$K \leftarrow_{\text{R}} \mathsf{stE.Gen}()$
$(\text{ST}_e^0, \text{ST}_d^0) \leftarrow_{\text{R}} \mathsf{stE.Init}()$
$i \leftarrow 0; j \leftarrow 0; \mathsf{phase} \leftarrow 0$
$b \leftarrow_{\text{R}} \{0,1\}$
$b' \leftarrow_{\text{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}$
Ret $(b' = b)$

procedure $\mathsf{Enc}(\ell, \text{H}, m_0, m_1)$:
$i \leftarrow i + 1$
$(c^0, \text{ST}_e^0) \leftarrow_{\text{R}} \mathsf{stE.Enc}(K, \ell, \text{H}, m_0, \text{ST}_e)$
$(c^1, \text{ST}_e^1) \leftarrow_{\text{R}} \mathsf{stE.Enc}(K, \ell, \text{H}, m_1, \text{ST}_e)$
If $c_i^0 = \perp$ or $c_i^1 = \perp$ then return $\perp$
Set $c_i = c^b$, $\text{H}_i = \text{H}$ and $\text{ST}_e = \text{ST}_e^b$
Ret c_i

procedure $\mathsf{Dec}(\text{H}, C)$:
If $b = 0$ then Ret $\perp$
$j \leftarrow j + 1$
$(m, \text{ST}_d) \leftarrow \mathsf{stE.Dec}(K, \text{H}, c, \text{ST}_d)$
If $j > i$ or $c \neq c_j$ or $\text{H} \neq \text{H}_j$ then
    $\mathsf{phase} \leftarrow 1$
If $\mathsf{phase} = 1$ then Ret m
Ret $\perp$

Fig. 6. The sLHAE security game.

We associate to adversary $\mathcal{A}$ and stateful AEAD scheme stE the advantage measure

$$\mathbf{Adv}^{\text{lh-st-ae}}_{\mathsf{stE}}(\mathcal{A}) = \Pr[\,\text{sLHAE1}_{\mathsf{stE}} \Rightarrow \mathsf{true}\,] - \Pr[\,\text{sLHAE0}_{\mathsf{stE}} \Rightarrow \mathsf{true}\,]$$

where $\text{sLHAE1}_{\mathsf{stE}}$ (resp. $\text{sLHAE0}_{\mathsf{stE}}$) is the game defined by setting $b = 1$ (resp. $b = 0$) in game $\text{sLHAE}_{\mathsf{stE}}$ shown in Figure 6. We say that stE is sLHAE-secure when the advantage of all adversaries consuming "reasonable" resources is "small" (where "reasonable" and "small" can be quantified in a concrete security setting).

Notice that when $b = 0$ in this game, the adversary is given access to an encryption oracle for m_0 and a decryption oracle that always returns $\perp$, while if $b = 1$, then the adversary is given access to an encryption oracle for m_1 but a decryption oracle that correctly decrypts decryption queries after $\mathsf{phase} \leftarrow 1$ (and returns $\perp$ otherwise, in order to suppress outputs from the decryption oracle allowing trivial wins for the adversary). The adversary's task is to distinguish these two views. This game captures both the desired confidentiality and integrity properties of a stateful AEAD scheme. Notice that we also modify the conditions under which $\mathsf{phase} \leftarrow 1$ from [42] so as to capture header integrity as well as ciphertext integrity. This gives a stronger and more natural definition than in [42].

It is shown in [42] that the sLHAE security notion is equivalent to the INT-sfCTXT integrity notion from [10] in combination with a stateful, length hiding IND-CPA security notion. A main result of [42] is that the encryption scheme MEE-TLS-CBC employed in the TLS Record Protocol is sLHAE-secure under reasonable hypotheses concerning the security of its MAC and block cipher components and the relative size of MAC tags and cipher blocks. Results from [35] can be used to show that when the TLS Record Protocol is instantiated using a stream cipher, then it is also sLHAE-secure (though without providing any length hiding, i.e. in the degenerate case where $|m_0| = |m_1|$ in all encryption queries).

B.2 PKE, KEMs and PRFs

A *public key encryption* (PKE) scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ consists of three polynomial-time algorithms (PTAs). Via $(\text{PK}, \text{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ the randomized key-generation algorithm produces public/secret keys for security parameter 1^λ; via $C \leftarrow \mathsf{Enc}(\text{PK}, M)$, the randomized encapsulation algorithm produces a ciphertext C for a message $M \in \mathcal{M}$; via $M \leftarrow \mathsf{Dec}(\text{SK}, C)$, the possessor of secret key SK decrypts ciphertext C to get back a message M or a special reject symbol $\perp$. For consistency, we require that

$$\Pr[\,C \leftarrow \mathsf{Enc}(\text{PK}, M); \mathsf{Dec}(\text{SK}, C) = M\,] = 1,$$

where the probability is taken over the choice $(\text{PK}, \text{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $M \leftarrow \mathcal{M}$ and the coins of all the algorithms in the expression above.

A *key encapsulation mechanism* (KEM) $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with key-space $\{0,1\}^\lambda$ consists of three PTAs. Via $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ the randomized key-generation algorithm produces public/secret keys for security parameter 1^λ; via $(\psi, K) \leftarrow \mathsf{Enc}(\mathrm{PK})$, the randomized encapsulation algorithm creates a uniformly distributed symmetric key $K \in \{0,1\}^\lambda$, together with a ciphertext ψ; via $K \leftarrow \mathsf{Dec}(\mathrm{SK}, \psi)$, the possessor of secret key SK decrypts ciphertext C to get back a key K which is an element in $\{0,1\}^\lambda$ or a special reject symbol $\perp$. For consistency, we require that

$$\Pr[\,(\psi, K) \leftarrow \mathsf{Enc}(\mathrm{PK}); \mathsf{Dec}(\mathrm{SK}, \psi) = K\,] = 1,$$

where the probability is taken over the choice $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and the coins of all the algorithms in the expression above.

A *labeled key encapsulation mechanism* (LKEM) $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is like a KEM, except both Enc and Dec take an additional input a label $L \in \{0,1\}^*$. For consistency, we require that for all labels $L \in \{0,1\}^*$,

$$\Pr[\,(\psi, K) \leftarrow \mathsf{Enc}(\mathrm{PK}, L); \mathsf{Dec}(\mathrm{SK}, L, \psi) = K\,] = 1,$$

where the probability is taken over the choice $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and the coins of all the algorithms in the expression above.

Pseudorandom functions. A *pseudorandom function* (PRF) with key space $\{0,1\}^\lambda$ and input space $\{0,1\}^*$ is a single deterministic polynomial-time algorithm PRF. On input a key K and a string x, the algorithm outputs a value $\mathsf{PRF}(K, x) \in \{0,1\}^\lambda$. For a stateful adversary, we define the advantage function

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A}) := \Pr\left[\, b' = b : \begin{array}{l} K \leftarrow_{\mathrm{R}} \{0,1\}^\lambda; \\ x \leftarrow \mathcal{A}^{\mathsf{PRF}(K,\cdot)}(1^\lambda); \\ K_0 := \mathsf{PRF}(K, x); K_1 \leftarrow_{\mathrm{R}} \{0,1\}^\lambda; b \leftarrow_{\mathrm{R}} \{0,1\}; \\ b' \leftarrow \mathcal{A}^{\mathsf{PRF}(K,\cdot)}(K_b) \end{array} \right] - \frac{1}{2}$$

with the restriction that $\mathcal{A}$ never queries $\mathsf{PRF}(K, \cdot)$ on input x. A PRF is said to be *pseudorandom* if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{PRF}}(\mathcal{A})$ is a neligible function in λ.

Signature scheme. A *signature scheme* with message space $\{0,1\}^*$ consists of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{sig}, \mathsf{Ver})$. For a stateful adversary $\mathcal{A}$, we define the advantage function:

$$\mathbf{Adv}^{\mathrm{sig}}_{\mathsf{sig}}(\mathcal{A}) := \Pr\left[\mathsf{Ver}(\mathrm{PK}, M^*, \sigma^*) = 1 : \begin{array}{l} (\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(1^k); \\ (M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{sig}(\mathrm{SK},\cdot)}(\mathrm{VK}) \end{array}\right]$$

with the restriction that $\mathcal{A}$ never made a query to $\mathsf{sig}(\mathrm{SK}, \cdot)$ on the message M^*. A signature scheme is said to be *existentially unforgeable* if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathrm{sig}}_{\mathsf{sig}}(\mathcal{A})$ is a negligible function in λ.

C On the Necessity of the PRF-ODH Assumption

The proof of TLS-DH assumes the key derivation function Kdf to satisfy the PRF-ODH assumption (Section 7.2). Here we prove that solely assuming Kdf to be a (regular) PRF is not sufficient to guarantee the security of TLS-DH (confirming the intuitive necessity of PRF-ODH from [31]). We show that if a

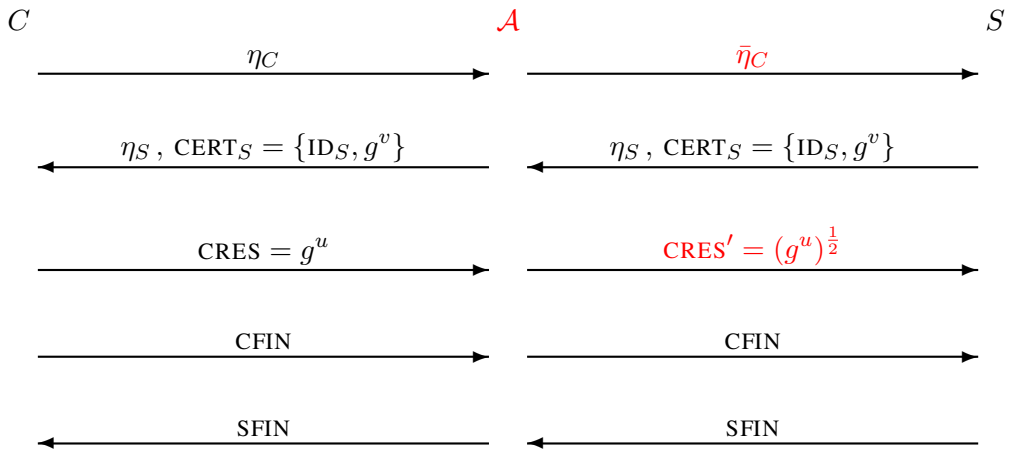

Fig. 7. Man-in-the-Middle against TLS-DH with a non-PRF-ODH Kdf

square-resistant PRF exists (as defined next and, in particular, implied by the existence of a PRF-ODH function) then there is a PRF that when used as Kdf in TLS-DH results in an insecure protocol. The attack is shown in Figure 7: The attacker replaces the client's nonce η_C with the same value but with its least significant bit flipped (the changed nonce is denoted by $\bar{\eta}_C$). Then in the CRES message, the attacker changes the client's value g^u to the value $(g^u)^{1/2}$. The rest of the messages are relayed between C and S unchanged. We will show that for a carefully drafted pseudorandom Kdf, with probability $1/4$, both C and S accept their respective sessions and derive the same application keys even though the sessions are non-matching due to the different nonces. This violates both channel security and server authentication. Here are the details.

Definition (square-resistant PRF). A PRF f with keys over a group G is called *square-resistant* if f_k is indistinguishable from random even when the attacker is given an oracle to f_{k^2} (this can be formalized in the setting of related-key security from Bellare et al. [8]).

Note: There are other forms of related key security that can be used here, squaring is an example. Heuristically, standard PRFs such as HMAC could be conjectured to be square-resistant but it would be interesting to show explicit examples of such functions (e.g., the Naor-Reingold scheme [40] or its variants may be good candidates as well as some pairing-based schemes). Interestingly, note that a PRF-ODH function is square-resistant which means that the existence of a PRF-ODH function simultaneously implies a secure instantiation of TLS-DH as well as the the existence of a PRF for which the protocol is insecure.

Assumptions. The analysis of the above attack uses the following two assumptions.

1. There exists a square-resistant PRF with keys over TLS DH group G (see above definition).
2. The identity function is used as the collision resistant hash function H in the computation of the Finished messages. (This means that if one is to prove TLS-DH with a regular PRF one would need to make assumptions on the function H beyond its collision resistance, e.g., modeling H as a random oracle.)

Definition of the function Kdf. Kdf is used in TLS to derive the master secret MS from a pre-master secret PMS as follows: $\text{MS} = \text{Kdf}(\text{PMS}, \texttt{`master secret'} \| \eta_c \| \eta_S)$ where $\text{PMS} = g^{uv}$.

Notation: We use x to denote the input to Kdf in the computation of MS and use $\bar{x}$ to denote a string equal to x except that the bit corresponding to the least significant bit position of η_C is flipped. We say that η_C is even if its least significant bit is 0 and say it is odd otherwise. For any function f we define f^* to be identical to f except that the least significant bit of the output is flipped.

Given a square-resistant PRF f we define Kdf as follows:

- For input x with even η_C, define $\mathsf{Kdf}(z, x) = f(z, x)$.
- For input x with odd η_C, define $\mathsf{Kdf}(z, x) = f^*(z^2, \bar{x})$.

Claim: The function Kdf is pseudorandom (follows from square-resistance of f).

Claim: For x with odd η_C, $\mathsf{Kdf}(z, x) = \mathsf{Kdf}^*(z^2, \bar{x})$ (both sides equal $f^*(z^2, \bar{x})$).

Corollary: For x with odd η_C:

$$\mathsf{Kdf}(g^{uv/2}, x) = \mathsf{Kdf}^*(g^{uv}, \bar{x}). \tag{1}$$

Definition of the function PRF. The TLS PRF is keyed with MS and used to derive the following values

- AKEY $= \mathsf{PRF}(\text{MS}, 0\|\eta_C\|\eta_S)$;
- UCFIN $= \mathsf{PRF}(\text{MS}, 1\|\eta_C\|\eta_S\|\text{CERT}_S\|\text{CRES})$;
- USFIN $= \mathsf{PRF}(\text{MS}, 2\|\eta_C\|\eta_S\|\text{CERT}_S\|\text{CRES}\|\text{UCFIN})$;

where the prefixes $0, 1, 2$ stand for TLS-defined labels (see Appendix B). In addition, the values CFIN, SFIN are obtained by encrypting UCFIN, USFIN, respectively, with the key AKEY.

Given a pseudorandom function F acting on inputs as the above PRF and with keys that are one bit shorter than MS, we define the pseudorandom function PRF as follows:

- If the least significant bit of MS is 0 then define for any input x, $\mathsf{PRF}(\text{MS}, x) = F(\text{MS}', x)$ where MS$'$ denotes the key MS with its last bit deleted;
- If the least significant bit of MS is 1 then define $\mathsf{PRF}(\text{MS}, x) = F(\text{MS}', x')$ where MS$'$ is defined as above and x' is defined as follows: the least significant bit of η_C is flipped and if CRES is included in x then in x' it is replaced with CRES^2 (recall that CRES is anelement in the group G).

Claim: The function PRF is pseudorandom (follows from the pseudorandomness of F and the fact that the transformation from x to x' is injective).

The values computed by C and S in the attack. The view of C has η_C as the client's nonce, $\text{CRES} = g^u$, and $\text{PMS} = g^{uv}$. The view of S has $\bar{\eta}_C$ as the client's nonce, $\text{CRES} = g^{u/2}$, and $\text{PMS} = g^{uv/2}$. We consider the case of even η_C (i.e., its lsb = 0) which occurs with probability 1/2 over the client's random choices, and denote by MS_C, MS_S, the values of MS derived by C and S, respectively. We have

- $\text{MS}_C = \mathsf{Kdf}(g^{uv}, \texttt{`master secret'}\|\eta_c\|\eta_S)$,
- $\text{MS}_S = \mathsf{Kdf}(g^{uv/2}, \texttt{`master secret'}\|\bar{\eta}_c\|\eta_S)$.

By Equation 1 we have that MS_C and MS_S are equal except for their last bit. Let m be the common prefix of MS_C, MS_S and assume MS_C ends with 0, i.e., we have $\text{MS}_C = m\|0$ and $\text{MS}_S = m\|1$.

Consider now the computation of UCFIN by the client and server and denote the computed values by these parties as UCFIN_C an UCFIN_S, respectively. We have:

- $\text{UCFIN}_C = \mathsf{PRF}(\text{MS}_C, 1\|\eta_C\|\eta_S\|\text{CERT}_S\|g^{uv})$;
- $\text{UCFIN}_S = \mathsf{PRF}(\text{MS}_S, 1\|\bar{\eta}_C\|\eta_S\|\text{CERT}_S\|g^{uv/2})$.

These two values are the same as they both equal $F(m, 1\|\eta_C\|\eta_S\|\text{CERT}_S\|g^{uv})$. Similarly, we have:

- $\text{AKEY}_C = \mathsf{PRF}(\text{MS}_C, 0\|\eta_C\|\eta_S) = F(m, 0\|\eta_C\|\eta_S) = \mathsf{PRF}(\text{MS}_S, 0\|\bar{\eta}_C\|\eta_S) = \text{AKEY}_S$;
- $\text{USFIN}_C = \mathsf{PRF}(\text{MS}_C, 1\|\eta_C\|\eta_S\|\text{CERT}_S\|g^{uv}\|\text{UCFIN}_C) = F(m, 1\|\eta_C\|\eta_S\|\text{CERT}_S\|g^{uv}\|\text{UCFIN}_C)$ $= \mathsf{PRF}(\text{MS}_S, 1\|\bar{\eta}_C\|\eta_S\|\text{CERT}_S\|g^{uv/2}\|\text{UCFIN}_S) = \text{USFIN}_S$

Thus, since the UCFIN, USFIN values computed by both parties are the same and so is the value of AKEY, then also CFIN and SFIN as computed by each party are the same. It follows that when both η_C and MS_C have a lsb of 0 (which happens with probability 1/4), both C and S accept the run of the protocol in Figure 7 as valid. In particular, they end with same application keys but in non-matching sessions (since the client nonce and CRES are different in the views of C and S). This breaks channel security as well as server authentication.