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. 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.

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 [20, 21], as well as related procotols like DTLS [31, 37]. In short, TLS is one of the most important real-world deployments of cryptography.

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. 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

* Supported by EPSRC Leadership Fellowship EP/H005455/1
** Supported by NSF CAREER Award CNS-1237429

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, [41, 12, 36, 28, 25, 40, 16, 27, 5, 32, 22, 6, 33, 18, 35, 9, 2, 24, 30, 19, 13, 3, 10, 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 [28] as well as Paterson, Ristenpart and Shrimpton [35]. These works demonstrated that, when carefully implemented to avoid timing and other attacks like those in [40, 16, 3], the stream-cipher and CBC encryption modes in the TLS Record Layer achieve the security notion of authenticated encryption; in fact, [35] 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 [8].

Several prior works [33, 25] 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 [33] analyze the "Truncated TLS Handshake Protocol", where the `Finished` messages are not encrypted by the application key. An important feature of [33] 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. [33] 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 [33] is developed further in recent work by Brzuska et al. [13], 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. [24] 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. [36, 22, 9] and on security analysis of a TLS implementation via type-checking [10]. Works on simulation-based definitions and designs for key agreement and secure channel protocols include [39, 14, 15].

TLS-RSA. As noted earlier, the most commonly deployed mode of TLS, namely TLS-RSA, uses RSA PKCS #1v1.5 encryption [26]. 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 [12], 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. [25]: 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. [24], we focus on the *cryptographic core* of TLS (see the full version [29] 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 [25] 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 [23] 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

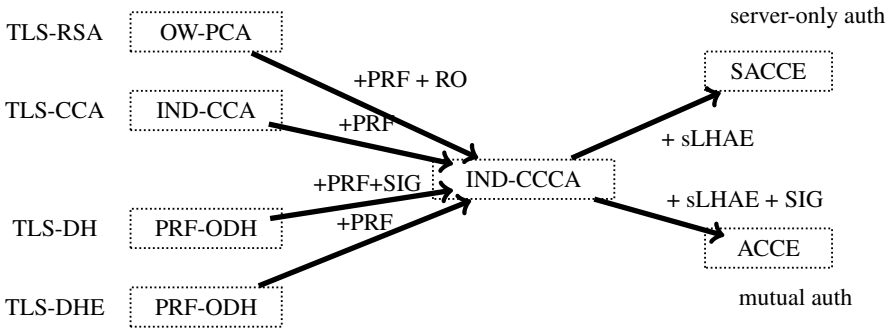

Fig. 1. Summary of our results.

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 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).

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 [25] and which we present in Section 5.2. We refer to [25] 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 [25]. 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, for example RSA-OAEP, 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 [24].[4] The PRF-ODH assumption rules out some potential related-key attacks on the

[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 [24].

Kdf function that would render the protocol insecure. In the full version [29] 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 [24].

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 [17], 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 [24]. We highlight several salient properties of our model, and defer a detailed justification and discussion to the full version [29]:

- 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 [12, 25]. 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 [12]. 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

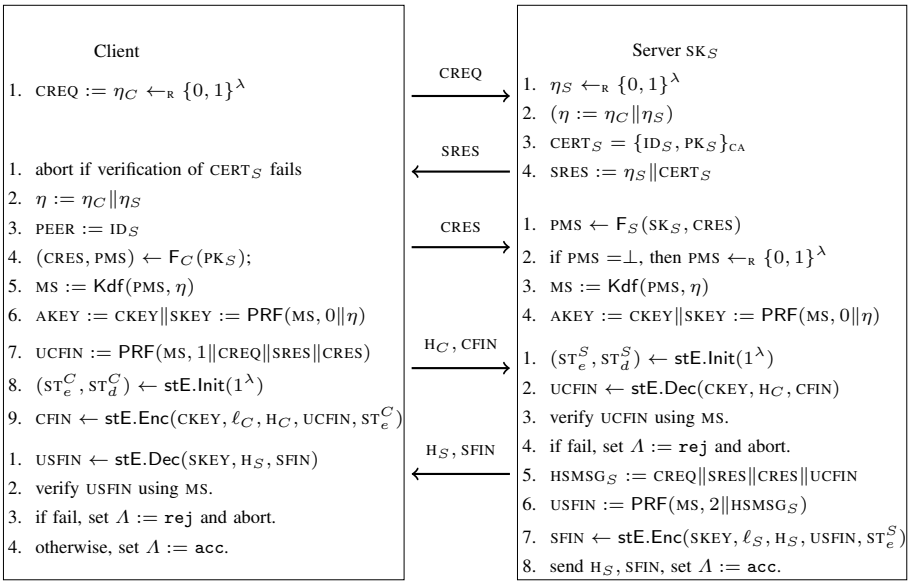

Fig. 2. Basic Generic TLS Handshake Protocol Parameterized by $(\mathsf{KeyGen}, \mathsf{F}_C, \mathsf{F}_S)$

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 6. 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 [17, 11]; 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 [17, 11], 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.
– 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 [24], we model the TLS Record Protocol via a stateful AEAD scheme stE which we will assume to be sLHAE-secure in the sense of [35]. For further details, see the full version [29].

3 Authenticated and Confidential Channel Establishment (ACCE)

We begin with the definition of *authenticated and confidential channel establishment (ACCE)* from [24, 8]. 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 protocol. An ACCE protocol is a protocol executed between two parties, a client and a server. In the original description [24], 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 $\textsc{akey}$. Then, encrypted and authenticated data is transmitted using a symmetric encryption scheme with the application key $\textsc{akey}$. More specifically, $\textsc{akey}$ is parsed as $\textsc{ckey}||\textsc{skey}$, the client uses $\textsc{ckey}$ in a stateful AEAD scheme stE to send data to the server, and the server uses $\textsc{skey}$ 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 $\textsc{peer}$ 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 $\textsc{akey}$ and executed $\mathsf{stE.Init}$. TLS meets this requirement.

3.1 Execution environment

Protocol entities. Following [24, 8], 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. Moreover, each $P \in \mathcal{S}$ (the servers) have a *unique* key pair $(\textsc{pk}_P, \textsc{sk}_P)$, an identity $\textsc{id}_P \in \{0,1\}^\lambda$ along with a certificate $\textsc{cert}_P := (\textsc{id}_P, \textsc{pk}_P)_{\textsc{ca}}$ signed by a certification authority $\textsc{ca}$. We also assume that all the parties in $\mathcal{S}$ have distinct identities.

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 game – 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:

- $\Lambda \in \{\emptyset, \mathtt{acc}, \mathtt{rej}\}$;
- $\text{AKEY} = \text{CKEY} \| \text{SKEY} \in \{0,1\}^{2\lambda}$, where $\{0,1\}^{2\lambda}$ is the application key space of the protocol;
- 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 $(\Lambda, \text{AKEY}, \text{PEER}) = (\emptyset, \emptyset, \emptyset)$, 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.

$\mathsf{Reveal}(\pi_i^P)$: 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 $\Lambda = \mathtt{acc}$.

$\mathsf{Encrypt}(\pi_i^P, \ell, \text{H}, m_0, m_1)$: if π_i^P has *not* reached state $\Lambda = \mathtt{acc}$, this oracle returns $\perp$. Otherwise, prior to reaching state $\mathtt{acc}$, π_i^P has (by assumption) computed AKEY and run the $\mathsf{stE.Init}$ 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 $\text{CKEY} \| \text{SKEY}$, and adds the 5-tuple $(\text{ST}_e, \text{ST}_d, b_i^P, \text{CKEY}, \text{SKEY})$ 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 $\mathsf{stE.Enc}$ and key $K = \text{CKEY}$ (if $P \in \mathcal{C}$) or key $K = \text{SKEY}$ (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.

$\mathsf{Decrypt}(\pi_i^P, \text{H}, c)$: 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 $P \in \mathcal{S}$). 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 $\mathsf{stE.Dec}$ 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.

Certificate authority. We assume that there is a single certificate authority (CA), which uses a secure signature scheme casig and its 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

$\mathsf{Encrypt}(\pi_i^P, \ell, \mathrm{H}, m_0, m_1)$:
$u \leftarrow u + 1$
$(c^0, \mathrm{ST}_e^0) \leftarrow_{\mathrm{R}} \mathsf{stE.Enc}(K, \ell, \mathrm{H}, m_0, \mathrm{ST}_e)$
$(c^1, \mathrm{ST}_e^1) \leftarrow_{\mathrm{R}} \mathsf{stE.Enc}(K, \ell, \mathrm{H}, m_1, \mathrm{ST}_e)$
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.

provide a certificate $\mathrm{CERT}_S := (\mathrm{ID}_S, \mathrm{PK}_S)_{\mathrm{CA}}$. We also allow the adversary access to the CA to register any number of parties, not in the set $\mathcal{S}$, with any public key of the adversary's choice.

Matching conversations. We consider a definition of matching conversation which is specific to TLS (and differs from the one in [24]):

Definition 1 (Matching conversations). *We say that π_i^P has a* matching conversation *with $\pi_j^{P'}$ if (i) either $P \in \mathcal{C}$ and $P' \in \mathcal{S}$, or $P \in \mathcal{S}$ and $P' \in \mathcal{C}$; and (ii) π_i^P accepts; and (iii) the transcripts at both π_i^P and $\pi_j^{P'}$ begin with the same three messages* $(\mathrm{CREQ}, \mathrm{SRES}, \mathrm{CRES})$.

Remark 1. Defining matching conversations as above means that we may treat the vector $(\mathrm{CREQ}, \mathrm{SRES}, \mathrm{CRES})$ 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 keys. 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, then we require that they have the 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, 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 [35]. We consider a game played between the adversary $\mathcal{A}$ and a challenger. This game is obtained by adapting [24, 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 $\{\text{ID}_S : S \in \mathcal{S}\}$.

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.)
  $\mathbf{Adv}_{\Pi}^{\text{sacce}-\text{sa}}(\mathcal{A})$ 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 $\text{PEER} = \text{ID}_S$, 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.)
  $\mathbf{Adv}_{\Pi}^{\text{sacce}-\text{ae}}(\mathcal{A})$ 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: (i) π_i^P reaches an accept state; (ii) 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; and (iii) $P \in \mathcal{C}$.

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 $\mathbf{Adv}_{\Pi}^{\text{sacce}-\text{sa}}(\mathcal{A})$ and $\mathbf{Adv}_{\Pi}^{\text{sacce}-\text{ae}}(\mathcal{A})$ are a negligible function of the security parameter λ.*

4 From CCCA KEM Security to SACCE Security of TLS

In this section we state 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 [23]:

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 2 (comparison with [23]). We point out the differences between our formulation and that in [23]. First, we consider a setting with labels. Second, in [23], 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 [25], we describe a labeled KEM which is extracted from the TLS protocol. Unlike [25] 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.

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 $\mathsf{tls.Enc}, \mathsf{tls.Dec}$ are as below.

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

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

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

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

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

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

where $\mathsf{PCA}(\text{SK}, \cdot, \cdot)$ is the oracle that takes as input (K, ψ) with $K \neq \perp$ and outputs 1 if $\mathsf{Dec}(\text{SK}, \psi) = K$ 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 $\mathbf{Adv}^{\text{ow-pca}}_{\mathsf{kem}}(\mathcal{A})$ is a negligible function in λ.

5.1 IND-CCCA from OW-PCA

The following lemma is similar to that in [25, Theorem 3], with some significant differences: (i) the KEM key in [25, Theorem 3] is AKEY and the ciphertext is $\text{CRES} \| \text{UCFIN}$, whereas our KEM key is $\text{AKEY} \| \text{USFIN} \| \text{UCFIN}$ and the ciphertext is simply CRES; (ii) [25] models PRF (referred to as h_s and h_z therein) also as a random oracle; (iii) [25] proves (labeled) IND-CCA security and does not consider encryption of UCFIN.

Lemma 1 (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

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

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

5.2 Implications for TLS-RSA

TLS-RSA KEM. The definition of the TLS-RSA KEM follows from the RSA PKCS #1v1.5 standard [26] adopted in TLS. Building on [25] 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 $(\mathrm{PK}, \mathrm{SK}) := ((N, e), d)$ 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 $P \leftarrow_{\mathrm{R}} \{0,1\}^{\lambda_1 - \lambda_0 - 24}$ 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 [26]. 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 [25, Theorem 1] via a reduction to an RSA-like assumption, known as "partial-domain RSA with decision oracle". The latter, given in [25, 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 [25, 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.[5]

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 [25] 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

[5] 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.

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 [25] over 10 years ago.

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

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-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, $\mathsf{dh.tlskem}$, that is IND-CCCA secure. Then, by Theorem 1 we conclude that TLS-DH is a secure SACCE protocol. Finally, we apply these results to TLS-DHE, namely, when both client and server provide ephemeral DH keys. Here, we merely provide a summary of our results; details can be found in the full version [29].

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

We show the security of TLS-DH in the standard model based on the PRF-ODH assumption on the function Kdf. This assumption is an adaptation of the Oracle Diffie-Hellman (ODH) assumption [1] to the PRF setting and was introduced in [24] for their proof of TLS-DHE. For the proof of TLS-DH we need the multi-query version of the assumption while for TLS-DHE the single-query case is sufficient, as in [24]. In [29] we describe the assumption and also prove its necessity by constructing secure pseudorandom functions for which PRF-ODH does not hold and with which TLS-DH violates ACCE security.

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 essentially 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 secure signature scheme, and sig_S is a signature produced by S under the corresponding signature key.

We note that [24] provided a specialized proof of TLS-DHE with client authentication. We obtain a proof for that particular case in Section 7 (Corollary 1). However, while [24] show forward security we do not include this property in our general treatment as it is not achieved by any of the other TLS modes.

7 The TLS Handshake Protocol with Mutual Authentication

In the full version [29] we augment server-only authentication, the SACCE model, with client authentication to obtain mutual authentication. The resultant model, ACCE, is mostly the same as in [24] except for the definition of matching sessions presented in Section 3. In this setting, the TLS client possesses a signature key (with a corresponding certificate CERT_C) which it uses to authenticate to the server (by computing a signature on the session transcript up to the CRES message). In this case both the definition of matching conversations and the TLS LKEM are augmented with CERT_C.

We first extend Theorem 1 to the case of mutual authentication, namely, showing that if tlskem is IND-CCCA secure and stE is sLHAE-secure then TLS with client and server authentication is ACCE-secure. Then we immediately obtain the following powerful corollary:

Corollary 1. *TLS-RSA, TLS-CCA, TLS-DH and TLS-DHE where the client is authenticated using a secure signature scheme are all ACCE secure under the same assumptions stated for the SACCE security of these modes.*

8 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. We summarise these issues here, expanding on them in [29].

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. 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. It also makes formal security analysis of the protocol easier, which in turn increases the likelihood that this analysis will be correct and applicable to the actual protocol under study.

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. Rather than moving to a CCA-secure implementation of the encryption function (e.g., via RSA-OAEP), the TLS community responded to Bleichenbacher's attack by keeping RSA PKCS#1v1.5 as the default implementation but disabling the error message in case of a decryption error. We stress that our proof of security for TLS-RSA relies crucially on there being no side channel that would reveal the existence of decryption failures to the attacker. While the TLS specification now takes care to avoid some explicit forms of leaking this information, implementations may still find ways of leaking it. 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.

The fragility of the TLS-RSA design is further illustrated by the somewhat "accidental" nature of its security. The security of this mode relies crucially on the fact that the client's `Finished` message, CFIN, is sent immediately after 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.

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 [18], the short MAC attack [35], 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.
- 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.

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 attacks from [19, 30, 38] are covered by our analysis.

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 P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.

[9] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Verified cryptographic implementations for TLS. *ACM Trans. Inf. Syst. Secur.*, 15(1):3, 2012.

[10] 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/.

[11] 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.

[12] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *CRYPTO*, pages 1–12, 1998.

[13] 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.

[14] 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.

[15] 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.

[16] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *CRYPTO*, pages 583–599, 2003.

[17] 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.

[18] T. Duong and J. Rizzo. Here come the $\oplus$ Ninjas. Unpublished manuscript, 2011.

[19] T. Duong and J. Rizzo. The CRIME attack. Presentation at ekoparty Security Conference, 2012. URL http://www.ekoparty.org/eng/2012/juliano-rizzo.php.

[20] 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.

[21] 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.

[22] 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.

[23] D. Hofheinz and E. Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*, pages 553–571, 2007.

[24] 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.

[25] J. Jonsson and B. S. Kaliski Jr. On the security of RSA encryption in TLS. In *CRYPTO*, pages 127–142, 2002.

[26] B. Kaliski. PKCS#1: RSA Encryption Version 1.5, Mar. 1998. URL http://www.rfc-editor.org/rfc/rfc2313.txt.

[27] V. Klíma, O. Pokorný, and T. Rosa. Attacking RSA-based sessions in SSL/TLS. In *CHES*, pages 426–440, 2003.

[28] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *CRYPTO*, pages 310–331, 2001.

[29] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. Cryptology ePrint Archive, Report 2013/339, 2013. Full version of this paper.

[30] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In *ACM CCS*, pages 62–72, 2012.

[31] 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.

[32] B. Moeller. Security of CBC ciphersuites in SSL/TLS: Problems and countermeasures. Unpublished manuscript, May 2004. http://www.openssl.org/~bodo/tls-cbc.txt.

[33] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In *ASIACRYPT*, pages 55–73, 2008.

[34] T. Okamoto and D. Pointcheval. REACT: Rapid enhanced-security asymmetric cryptosystem transform. In *CT-RSA*, pages 159–175, 2001.

[35] 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.

[36] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[37] E. Rescorla and N. Modadugu. Datagram Transport Layer Security, Apr. 2006. URL http://www.rfc-editor.org/rfc/rfc4347.txt.

[38] 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.

[39] V. Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. http://eprint.iacr.org/.

[40] S. Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In *EUROCRYPT*, pages 534–546, 2002.

[41] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.