Universally Composable Security Analysis of TLS— Secure Sessions with Handshake and Record Layer Protocols*

Sebastian Gajek[1], Mark Manulis[2], Olivier Pereira[2], Ahmad-Reza Sadeghi[1] and Jörg Schwenk[1]

[1]Ruhr University Bochum, Germany
[2]Université Catholique de Louvain, Belgium

Abstract

We present a security analysis of the complete TLS protocol in the Universal Composable security framework. This analysis evaluates the composition of key exchange functionalities realized by the TLS handshake with the message transmission of the TLS record layer to emulate secure communication sessions and is based on the adaption of the secure channel model from Canetti and Krawczyk to the setting where peer identities are not necessarily known prior the protocol invocation and may remain undisclosed. Our analysis shows that TLS, including the Diffie-Hellman and key transport suites in the uni-directional and bi-directional models of authentication, securely emulates secure communication sessions.

Keywords: Universal Composability, TLS/SSL, key exchange, secure sessions

*Corresponding author's email address: `sebastian.gajek@nds.rub.de`

Contents

1 Introduction

1.1 Motivation

The protocol framework of *Transport Layer Security (TLS)* [18] serves as fundamental primitive for WWW security and has fostered to the most valuable cryptographic protocol family in practice. The TLS protocol suites enable applications to communicate across a distributed network in a way that endpoint authentication and transmission privacy is guaranteed. Prominent examples include tunneling to create a virtual private network, protect Internet phone calls, and secure the rich facets of multi-party Internet applications, such as online banking, electronic commerce or federated identity management, just to name a few.

The main goal of this paper is to provide a rigorous and generic analysis of TLS's cryptographically relevant parts of the protocol framework, namely the handshake and record-layer protocols. Given the wide deployment of TLS and the fact that it has been designed as contemporary cryptography started to explore provable security, it is natural that this analysis is of high, practical interest. Since TLS has already been investigated with respect to certain cryptographic primitives and protocol abstractions (see below), a general belief is that the framework is secure. Yet, there is no security proof of the entire TLS protocol in a solid framework and a careful observation of TLS's subtleties in the various modes provided by the different cipher suites. However, such a proof would significantly contribute to the analysis of complex protocols executed on top of TLS.

Our analysis is carried out in the meanwhile classical model of Universally Composable (UC) security [5] which guarantees protocol security under general composition with arbitrary other protocols. This valuable property stimulated the search for universal protocol design techniques and their realizations [8, 10, 27, 30, 14, 15]. On the other hand, there are important impossibility results [8, 31] so that a security proof of TLS in this model is neither obvious nor trivial. Our work particularly continues the way of Canetti's and Krawczyk's consideration of the Σ-protocol underlying the signature based modes in IPSec [12] and their model to build up secure channels [13] in the UC model with the exception that instead of proving single modes, we utilize UC as technique to prove the complete protocol secure in a single proof. Applied to the analysis of TLS, it includes Diffie-Hellman and encrypted key transport in the uni- or bi-directional model of authentication which are part of the TLS handshake, and their emulation to build secure communication protocols realized by the additional TLS record layer.

The most relevant question is how to reduce the complexity of the proof. Is it possible to unitize TLS in meaningful protocol fragments such that the composition theorem allows for an efficient protocol reformulation in the hybrid model? That means, can we define ideal functionalities that capture the cryptographic task of some of its fragments and simply reuse these functionalities with the next fragment? Otherwise, a composite analysis would not make sense so that we could switch to stand-alone protocol proofs. Fortunately, we answer the questions in the positive. To this end, we introduce two ideal functionalities, dubbed the *universal key exchange* and *universal secure communication sessions*. The functionalities are "universal" in the sense that they emulate different key establishment methods and modes of authentication in a self-contained definition. In contrast with the formulation in the post-specified setting as used for the analysis of the Σ-protocol in [12] or more recently in [34], where the peer identities are disclosed during the protocol execution, in the responder authenticated setting the server identity is publicly known at the start. However, in TLS the client identity may remain undisclosed at the end of the protocol implying the anonymous uni-directional model of authentication which is of prime interest for anonymous user authentication [41]. In which case, the client reveals its own identity and authenticates using a password which is triggered over a higher-layer protocol on top of TLS (e.g. HTTPS, FTPS). These constructions constitute the layered Internet approach for designing network security protocols. However, the protocols significantly differ from universally composable password-based authenticated channels [10, 21]. Here, the adversary can attack the composed protocol (e.g. [39, 22, 32]). We show that the TLS framework including the different modes securely emulates the universal secure sessions functionality in the presence of non-adaptive adversaries. Our result can significantly simplify security proofs of higher-layer protocols by employing the composition theorem. We are not aware of any prior work that evaluates the essential composability property of TLS.

1.2 Related Work

Because of its eminent role the TLS framework has been repeatedly peer-reviewed. Schneier and Wagner [40] gave the first informal analysis in the core specification. Bleichenbacher [4] found some weaknesses in the PKCS#1 standard for RSA encryption as used with some SSL 3.0 handshake protocols.[1] Jonsson and Kaliski [29] showed that the encryption in the revised PKCS#1.5 standard is secure against chosen cipher attacks in the Random Oracle Model. Krawczyk [33] analyzed the composition of symmetric authentication and encryption to establish a secure communication channel with TLS record layer protocols and found some problems in the case of general composition. However, these do not apply to the standard cipher suites.

Apart from the analysis of some cryptographic primitives, a line of research addressed the analysis of *dedicated* TLS protocols on the basis of cryptographic abstractions to allow automated proof techniques. Paulson [38] gave an inductive analysis of a simplified version of TLS, using the theorem proving tool Isabelle. Mitchell, Shmatikov, and Stern [35] checked TLS, using the finite-state enumeration tool named Murphϕ. Ogata and Futatsugi [37] used the interactive theorem prover OTS/CafeObj to check a simplified version of the key transport handshake protocol through equational reasoning. He *et al.* [26] provided a proof of correctness of TLS in conjunction with the IEEE 802.11i wireless networking protocol, using the Protocol Composition Logic. The drawback these tool-supported approaches currently share is that the proofs are considerably simplified. They follow the *Dolev-Yao model* [19] which represents cryptography as term algebras and abstracts away the comprehensiveness of the adversary such that the proofs are not known to be cryptographically sound.

Very recently, Morrissey *et al.* [36] analyzed in an independent and yet unpublished work the modularity of a *TLS-related* handshake protocol in a game-based style. The handshake is not exactly conform with the core TLS specification [18] and considers not all protocol variants. Their work focuses on a generic proof of the iterated session key constructions. By contrast, our work is of independent interest and practical relevance. We investigate TLS's intrinsic compositional property which is to provide higher-layer protocols with some secure communication functionality. Furthermore, our work addresses the native handshake protocols and additionally the record layer protocols in different authentication models under the stronger security notion of universally composable security.

[1] Note that the attack exploited weaknesses of the PKCS#1 standard and not the TLS protocol.

1.3 Organization

The remaining sections are structured as follows. Section 2 clarifies notation and cryptographic building blocks. Section 3 reviews the Universal Composability Framework. Section 4 shortly introduces the TLS protocol family and describes the compositional proof idea. Section 5 is devoted to the TLS handshake subroutines we use throughout the analysis. Section 6 proves the full framework and Section 7 concludes.

2 Preliminaries

2.1 Notations

The protocols run between two players: a client and a server. A player may act as *initiator* I or *responder* R. By $P \in (I,R)$ we denote a pair of such players and by $\bar{P}$ the same pair but in the reverse order, i.e. (R, I). An anonymous player, i.e. a party whose identity is not known is denoted by $\perp$. We refer to the handshake protocol structure as π and the composition with the record-layer protocols as ρ. Additionally, we use different indices to capture the modes of authentication in ideal functionalities. We refer to a responder-only authenticated functionality as $\mathcal{F}^1$, i.e. a functionality where the responder authenticates to the initiator, but the initiator's identity remains unknown. Further, we denote an ideal functionality, where both players authenticate by $\mathcal{F}^2$, and a hybrid functionality of $\mathcal{F}^1$ and $\mathcal{F}^2$ by $\mathcal{F}^{(1,2)}$.

2.2 Cryptographic Building Blocks and their Constructions

The specification of TLS [18] uses several cryptographic primitives and mandates or recommends certain instantiations of them as described in the following:

An ASYMMETRIC ENCRYPTION SCHEME $(\mathtt{ENC}_{pk_R}(), \mathtt{DEC}_{sk_R}())$ for transporting the encrypted premaster secret which must be instantiated with the RSA-OAEP construction (known to provide indistinguishability under adaptive chosen ciphertext attacks [29] in the Random Oracle Model). In TLS handshake a private key sk_R is known to the responder R and its public key pk_R is signed by a Certification Authority (CA).

A DIGITAL SIGNATURE SCHEME $(\mathtt{SIG}_{sk}(), \mathtt{VER}_{vk}())$ for entity authentication which can be instantiated with DSA and RSA-PSS (the latter is known to provide weak existential unforgeability under chosen message attacks in the Random Oracle Model [28]). The players own a signing key sk and the respective verification vk is certified by a CA.

A MESSAGE AUTHENTICATION CODE function $\mathtt{HMAC}_k()$ from [2] and a SYMMETRIC ENCRYPTION SCHEME $(\mathtt{E}_k(), \mathtt{D}_k())$ which is recommended to be DES or 3DES in different modes and with different key lengths. The construction of symmetric authentication *with* encryption is known to provide weak unforgeability under chosen message attacks and indistinguishability under chosen plaintext attacks [33, 3].

A PSEUDO-RANDOM FUNCTION for the key derivation and confirmation, denote here by $\mathtt{PRF}_k()$. It is evaluated with seed k on an input string l_i, $i \in [1, 4]$ which is labeled with different publicly known space delimiters and two independently chosen random values, i.e. the nonces exchanged in the first protocol, or a function thereof. The specification defines a special construction based on HMAC combiners which has been recently proven to be a good randomness extractor [20].

3 The Universal Composability Security Framework

We give an overview of the UC security framework, referring the reader to [5] for a comprehensive description.

3.1 System Model

In the UC framework, Interactive Turing Machines (ITM) interact in two worlds. See Fig. 1. The real-world model comprises honest parties and the adversary $\mathcal{A}$. The parties run a protocol π in order to compute a cryptographic task. $\mathcal{A}$ controls the communication and potentially corrupts the parties. The ideal world includes "dummy" parties who interact with an ideal functionality $\mathcal{F}$, running the ideal protocol ϕ. The functionality $\mathcal{F}$ represents a trusted party that carries out the same cryptographic task. It simply obtains the inputs of all players and provides them with the desired outputs. The ideal-world adversary $\mathcal{S}$ (dubbed the *simulator*) is allowed to delay messages. However, is unable to gain knowledge of any inputs/outputs except the functionality $\mathcal{F}$ is willing to grant it. Intuitively, the ideal functionality captures the security requirements of a given cryptographic task we expect from the real-world protocol π and defines the adversarial corruption model we consider in that setting.

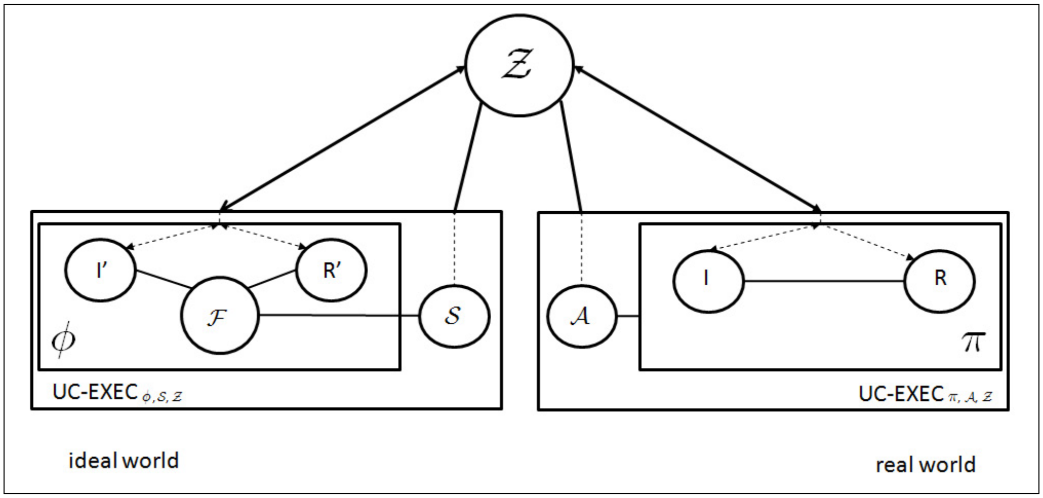

Figure 1: The Real World/Ideal World Paradigm in the UC Framework. In the real world, player I and R execute protocol π in front of adversary $\mathcal{A}$. In the ideal world, the dummy players I' and R' interact with the ideal Functionality $\mathcal{F}$ in presence of the simulator $\mathcal{S}$ to compute the same cryptographic task.

3.2 Security Definition

In the UC framework there exists an additional entity called the environment $\mathcal{Z}$. The environment plays the role of a "judge" who has to distinguish between the two worlds. Therefore, the environment feeds all parties with input, retrieves their outputs, and interacts with the adversary in an arbitrary way throughout the computation. The ideal-world adversary $\mathcal{S}$ does not perceive the message exchange between the real-world parties and has to simulate the interaction in order to mimic the behavior of $\mathcal{A}$. Then, security of protocol π is captured by the fact that every attack $\mathcal{A}$ mounts in the real world, $\mathcal{S}$ carries out in the ideal world. The protocol security is implied, since in the ideal world such attacks cannot be mounted. We have then that the outputs $\mathcal{Z}$ retrieved from the execution of ϕ with the dummy players and $\mathcal{S}$ and the execution of π with the real-world players and $\mathcal{A}$ are indistinguishably distributed. Here, indistinguishability means in this case computational indistinguishability ("$\approx$"). Informally, a protocol π is said to *securely emulate* an ideal-world protocol ϕ. In addition, a protocol π is said to *securely realize* a cryptographic task, if for any real-world adversary $\mathcal{A}$ that interacts with $\mathcal{Z}$ and real players running π, there exists an ideal-world simulator $\mathcal{S}$ that interacts with $\mathcal{Z}$, the ideal functionality $\mathcal{F}$, and the dummy players running the ideal protocol ϕ, so that no probabilistic polynomial time environment $\mathcal{Z}$ is able to distinguish whether it is interacting with the real-world $\mathcal{A}$ or the ideal-world adversary $\mathcal{S}$. A more general definition is:

Definition 1 *A protocol π* `UC-emulates` *protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for all environments $\mathcal{Z}$ that output only one bit:*

$$\texttt{UC}-\texttt{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \texttt{UC}-\texttt{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$$

A protocol π `UC-realizes` *an ideal functionality $\mathcal{F}$ if π UC-emulates the ideal protocol for $\mathcal{F}$.*

We sometimes abuse the notation and write $\texttt{UC}-\texttt{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \texttt{UC}-\texttt{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ to say that π UC-realizes an ideal functionality $\mathcal{F}$. It is easy to see that ideal protocol ϕ is the protocol that defines the communication between $\mathcal{F}$ and the dummy players that simply forward their inputs and outputs. This is equivalent to $\mathcal{F}$ bypassing the dummy players.

Relaxed UC Security The standard notion of UC security is a strong security definition and rules out the simulatability of some important, provably secure protocols. In order to make the restriction clear, we recall the example from [13]. Consider a two-move Diffie-Hellman protocol. Assume that a prime p and a generator g of a large subgroup of $\mathbb{Z}_p^*$ of prime order are given. The initiator fixes $x \overset{r}{\leftarrow} \mathbb{Z}_q$ and sends $\alpha = g^x$ to the responder. Upon reception the responder fixes $y \overset{r}{\leftarrow} \mathbb{Z}_q$ and sends $\beta = g^y$. Both players locally output g^{xy}. Simulating the two-move Diffie-Hellman protocol with access to a functionality, say $\mathcal{F}_{\text{KE}}^{\text{bad}}$, which independently fixes the shared key μ at random yields a view that allows the environment to distinguish the two worlds. To understand why, assume that the simulator comes up with the values α' and β'. Next, the environment instructs the adversary to corrupt the initiator before receiving the responder's answer. Then, the environment learns the random value fixed by $\mathcal{F}_{\text{KE}}^{\text{bad}}$ due to the output from the responder and the simulator has to come up with a value x' such that $\beta'^{x'} = \mu$. Since the values α' and β' are independent from μ, a value x' exists only with negligible probability.

To mitigate the limitations, a relaxation of the UC security definition has been proposed in [13] by providing the functionality with some help in form of a non-information oracle $\mathcal{N}$. The oracle outputs a value which is indistinguishable from a random value. More formally, let $\mathcal{N}$ be a polynomial time machine interactive Turing machine. Then $\mathcal{N}$ is a non-information oracle if no interactive Turing machine $\mathcal{M}$, having interacted with $\mathcal{N}$ on security parameter k, can distinguish with non-negligible probability between the local output of $\mathcal{N}$ and a value drawn uniformly from $\{0,1\}^k$. The purpose of the non-information oracle is to supply the simulator with auxiliary information to make the output from the simulation conform to the output from the functionality. The simulator interacts with the non-information oracle and receives the input for the simulation. If the adversary corrupts a player, then $\mathcal{N}$ discloses its current session state to the adversary, including the randomness (x, y) and its local output (g^{xy}).

Definition 2 *A protocol π is said to be* `relaxed UC-secure` *if there exists a non-information oracle $\mathcal{N}$ such that π securely realized $\mathcal{F}^{\mathcal{N}}$.*

In particular, realizing a key exchange functionality under the relaxed definition has been shown to be equivalent to the notion of SK-security. Informally, a key exchange protocol is said to be SK-secure, if (i) no adversary can force the partners of the session to output different session keys, and in addition (ii) guesses whether the output was the real session key or a random test value. The proof is given in [13].

3.3 Universal Composition

A key point of the UC framework is the composition theorem. It guarantees composition with arbitrary sets of parties. Consider a protocol ρ that operates in the $\mathcal{F}$-hybrid model. That is, parties interact in the normal way and in addition can invoke an arbitrary number of copies of the functionality $\mathcal{F}$. We call the invocation of $\mathcal{F}$ subroutine-respecting, if only ρ is permitted to receive the inputs and outputs of the ideal functionality. Then, the following holds.

Theorem 3 *Let π and ϕ be two subroutine-respecting polynomial-time protocols such that π UC-emulates ϕ. Then $\rho^{\pi/\phi}$ UC-emulates ρ for any polynomial-time protocol ρ.*

If π UC-emulates ϕ, we have that there is no $\mathcal{Z}$ that can distinguish with non-negligible probability between the players running π and players running ϕ in the presence of the adversary. The subroutine-respecting invocation ensures that the surrounding protocol ρ feeds π and ϕ in the same way so that the outputs are identical distributed. The composition theorem prevails that replacing the instance of π with an instance of ϕ does not change the behavior of ρ with respect to any polynomial-time adversary; we have a symmetry between the case that ρ interacts with π and ϕ in the presence of the adversary. The main attraction of the composition theorem follows from the fact that if ϕ UC-realizes $\mathcal{F}$ then the real-world protocol ρ can replace the invocation of subroutine π by calling the ideal functionality. The full proof is detailed in [7].

In some cases the universal composition operation would result in highly inefficient protocols. Consider a key exchange protocol that calls a signature subroutine for authenticating the keys. The universal composition theorem states that for each instance of the key exchange protocol a new instance of the signature module is invoked. Consequently, the subroutine would generate for each key exchange a new pair of signature and verification keys. It becomes more involved when the subroutine applies certified keys issued by a public authority where multiple players use the same key (as required in many cryptosystems to setup the protocol). The *composition theorem with joint state (JUC)* avoids this unnecessary complexity [16]. This operation is similar to universal composition except that multiple instances of a protocol can gain access to the same instance of a subroutine in order to benefit from a joint state (e.g. the signature key is the joint state).

Theorem 4 *Let $\mathcal{F}$ be an ideal functionality. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let $\hat{\rho}$ be protocol that securely realizes $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$. Then the composed protocol $\pi^{[\hat{\rho}]}$ emulates protocol π in the $\mathcal{F}$-hybrid model.*

The intuition behind the JUC theorem is as follows. Observe a protocol π in the $\mathcal{F}$-hybrid model. Next, observe a protocol $\hat{\rho}$ that realizes $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$. The functionality $\hat{\mathcal{F}}$ maintains multiple copies of $\mathcal{F}$. Technically, the multi-session extension is responsible for the invocation of an appropriate copy of $\mathcal{F}$. Upon invocation of $\hat{\mathcal{F}}$, the protocol participants perceive the cryptographic task of $\mathcal{F}$. Consequently, $\mathcal{Z}$'s view when interacting with the players running an instance of π in the $\hat{\rho}$-hybrid model is computationally indistinguishable from its view when interacting with π in the $\mathcal{F}$-hybrid model.

4 Transport Layer Security

4.1 TLS in a Nutshell

The standard TLS specification [18] comprises handshake, alert, change cipher spec, and record layer (sub)protocols. The *handshake protocol* is used to negotiate key material and cryptographic algorithms and the record layer protocol can then be applied to secure transmitted application data. The *change cipher spec protocol* consisting of one message triggers a change in the cryptographic parameters used by the record layer, while the *alert protocol* communicates error messages, whenever a failure during the handshake or message protection occurs. Thus, the essential cryptographic building blocks for TLS and target to the presented analysis are the handshake and record layer protocols.

Handshake and Record Layer The TLS handshake aims at the negotiation of a common secret called the *master secret* k_m which is in turn derived from the the previously established *premaster secret* k_p. The modularity of the handshake protocol is captured by the fact that different subroutines are applied to establish the premaster secret and derive the master secret while the remaining structure of the handshake is unchanged (see Fig. 2). TLS distinguishes among the following subroutines: encryption of the premaster secret using the server's public key (EKT); static (DHS) or ephemeral signed (DHE) Diffie-Hellman key exchange. Optionally, TLS allows for the client authentication via a signature over all received values `trscrpt` which can be verified using the public key with the client certificate. The master secret k_m is then used to derive up to four cryptographic keys for the record layer: two symmetric encryption keys k_e^P (including an initialization vector for the block-cipher based encryption), and two authentication keys k_a^P, where $P \in \{I, R\}$. Finally, client and server confirm the negotiated security parameters by exchanging their finished messages which are derived from k_m and protected via *authenticated encryption* by the record layer (i.e. MAC of the plaintext is used as input to the symmetric encryption). The same protection is then applied to the subsequent application data.

Remark. Note that an application message may be fragmented and compressed when processed by the record layer. Therefore, the record layer encodes sequence numbers into the fragments and maintains a counter in order to prevent disorder. Note also that a key feature of TLS is session resumption in order to reduce server-sided performance penalties. The client names an earlier session that it intends to continue; if the server agrees, the previous master secret is used with the new nonces to generate new key material for the record layer. Though not explicitly treated in our paper, it is easy to see that the security of the abbreviated handshake follows from our analysis of the full handshake.

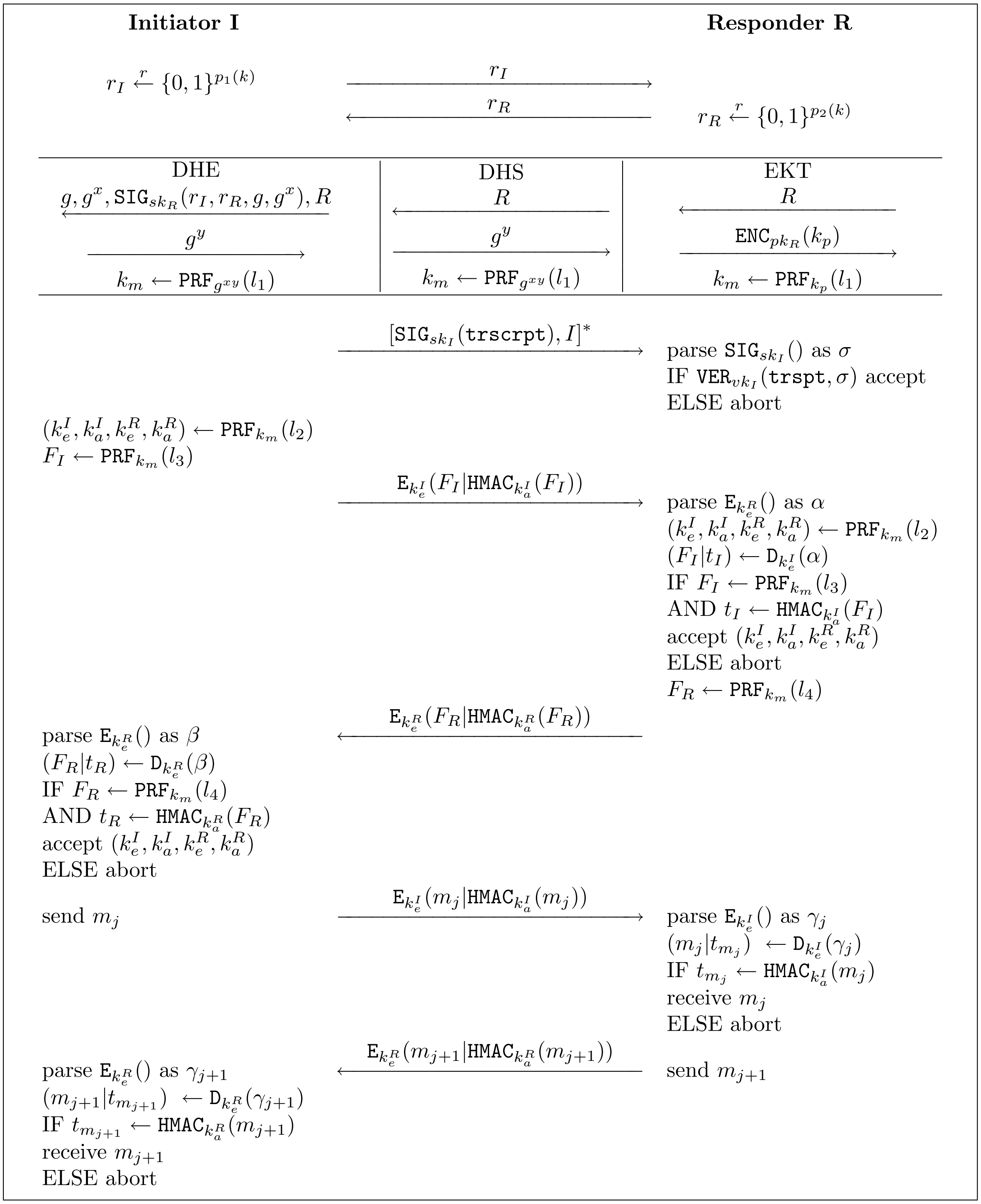
Initiator I
Responder R
$r_I \xleftarrow{r} \{0,1\}^{p_1(k)}$
r_I
r_R
$r_R \xleftarrow{r} \{0,1\}^{p_2(k)}$
DHE
$g, g^x, \mathtt{SIG}_{sk_R}(r_I, r_R, g, g^x), R$
g^y
$k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$
DHS
R
g^y
$k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$
EKT
R
$\mathtt{ENC}_{pk_R}(k_p)$
$k_m \leftarrow \mathtt{PRF}_{k_p}(l_1)$
$[\mathtt{SIG}_{sk_I}(\mathtt{trscrpt}), I]^*$
parse $\mathtt{SIG}_{sk_I}()$ as σ
IF $\mathtt{VER}_{vk_I}(\mathtt{trspt}, \sigma)$ accept
ELSE abort
$(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_{k_m}(l_2)$
$F_I \leftarrow \mathtt{PRF}_{k_m}(l_3)$
$\mathtt{E}_{k_e^I}(F_I|\mathtt{HMAC}_{k_a^I}(F_I))$
parse $\mathtt{E}_{k_e^R}()$ as α
$(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_{k_m}(l_2)$
$(F_I|t_I) \leftarrow \mathtt{D}_{k_e^I}(\alpha)$
IF $F_I \leftarrow \mathtt{PRF}_{k_m}(l_3)$
AND $t_I \leftarrow \mathtt{HMAC}_{k_a^I}(F_I)$
accept $(k_e^I, k_a^I, k_e^R, k_a^R)$
ELSE abort
$F_R \leftarrow \mathtt{PRF}_{k_m}(l_4)$
$\mathtt{E}_{k_e^R}(F_R|\mathtt{HMAC}_{k_a^R}(F_R))$
parse $\mathtt{E}_{k_e^R}()$ as β
$(F_R|t_R) \leftarrow \mathtt{D}_{k_e^R}(\beta)$
IF $F_R \leftarrow \mathtt{PRF}_{k_m}(l_4)$
AND $t_R \leftarrow \mathtt{HMAC}_{k_a^R}(F_R)$
accept $(k_e^I, k_a^I, k_e^R, k_a^R)$
ELSE abort
send m_j
$\mathtt{E}_{k_e^I}(m_j|\mathtt{HMAC}_{k_a^I}(m_j))$
parse $\mathtt{E}_{k_e^I}()$ as γ_j
$(m_j|t_{m_j}) \leftarrow \mathtt{D}_{k_e^I}(\gamma_j)$
IF $t_{m_j} \leftarrow \mathtt{HMAC}_{k_a^I}(m_j)$
receive m_j
ELSE abort
$\mathtt{E}_{k_e^R}(m_{j+1}|\mathtt{HMAC}_{k_a^R}(m_{j+1}))$
send m_{j+1}
parse $\mathtt{E}_{k_e^R}()$ as γ_{j+1}
$(m_{j+1}|t_{m_{j+1}}) \leftarrow \mathtt{D}_{k_e^R}(\gamma_{j+1})$
IF $t_{m_{j+1}} \leftarrow \mathtt{HMAC}_{k_a^R}(m_{j+1})$
receive m_{j+1}
ELSE abort

Figure 2: The TLS protocol including the different subroutines DHE, DHS, and EKT to establish the master secret k_m. (*) marks the optional client authentication message. Event 'abort' invokes the alert protocol with the respective error message; events 'send' and 'receive' trigger interfaces to the application layer.

4.2 Roadmap for the Modular Analysis of TLS

The structure of the TLS framework advocates its modular analysis. Intuitively, the handshake protocol captures the cryptographic task of key exchange and the composition with the record layer protocol emulates secure transfer of application messages. However, the straightforward idea to model the complete handshake protocol as ideal key exchange functionality in order to negotiate the session keys and compose it with the record layer protocol in order to realize a secure communication sessions functionality fails in general. The handshake protocol does not securely realize the ideal key exchange functionality since it uses the derived session keys to encrypt and authenticate finished messages. Thus, the environment can test the keys using the finished messages and tell the two worlds apart. See Appendix A for more discussions.

In our analysis we avoid this obstacle by devising a functionality $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ that emulates the handshake's subroutines to negotiate the master secret k_m (instead of a straight-line computation of the session keys). $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ captures the fact that two players receive a random key unless either player is corrupted. Next, we demonstrate that the subroutines DHE, DHS, and EKT securely realize $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ (Section 5). Our analysis is focused on responder-only and mutual authenticated communication which are the authentication modes supported by TLS (apart from anonymous Diffie-Hellman suites). Since TLS operates in a setting where the existence of a trusted third party in the sense of a Certificate Authority (CA) is required, we formalize the global setup assumption by formulating the real-world protocols in $\mathcal{F}$-hybrid models, utilizing the *certification functionality* $\mathcal{F}_{\mathrm{CERT}}$, *certified public key encryption functionality* $\mathcal{F}_{\mathrm{CPKE}}$, and *certificate authority functionality* $\mathcal{F}_{\mathrm{CA}}$, as presented in [6, 11].

The composition with these functionalities to a subroutine protocol is preserved by the JUC theorem. It is useful in the case of key exchange when multiple subroutine sessions have access to the same instance of functionalities $\mathcal{F}_{\mathrm{CERT}}$, $\mathcal{F}_{\mathrm{CPKE}}$, and $\mathcal{F}_{\mathrm{CA}}$, using the same key for authenticating multiple messages (i.e. the signature, encryption, and deposited key is the joint state, respectively). Finally, we make use of the composition theorem and specify the TLS protocol in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid model. We show that the reformulated TLS protocol securely realizes the ideal functionality for the secure communication sessions (Section 6).

5 Specification and Analysis of TLS Subroutines

We proceed with the specification and emulation of an ideal-world functionality which we henceforth call *universal key exchange* $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ that captures the requirements of the subroutines DHE, DHS, and EKT. These subroutines compute the master secret.

5.1 Universal Key Exchange Functionality

The key exchange functionality $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ is illustrated in Fig. 3. It mimics the cryptographic task that the players I and R agree upon a shared secret μ which is indistinguishable from an independently chosen value of the same length as long as a party is uncorrupted. There is a large body of literature that covers ideal key exchange functionalities (e.g. [5, 13, 11]). $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ is similar to these functionalities except for:

First, the players authenticate in a post-specified fashion, i.e. the environment invokes players with the session identifier SID and optionally their own identity. A player learns its peer identity

Functionality $\mathcal{F}_{\text{KE}}^{(1,2)}$

$\mathcal{F}_{\text{KE}}^{(1,2)}$ proceeds as follows when parameterized with security parameter k.

- Upon receiving an input ("establish-key", `SID`, ID_I) from some party, where $ID_I \in (\perp, I)$, record ID_I as initiator, and send a message ("establish-session", `SID`, ID_I) to the adversary. Upon receiving input ("establish-key", `SID`, R) from some other party, record R as responder, and send the message ("establish-key", `SID`, R) to the adversary.
- Upon receiving an answer ("impersonate", `SID`, $\tilde{\mu}$) from the adversary, do: If $ID_I{=}\perp$, record the adversary as initiator and send message ("Key", `SID`, $\perp$, $\tilde{\mu}$) to the responder. Else, ignore the message.
- Upon receiving an answer ("Key", `SID`, P, $\tilde{\mu}$) from the adversary, where P is either the initiator or responder, do: If neither initiator nor responder is corrupted, and there is no recorded key, fix μ uniformly from $\{0,1\}^k$. If either initiator or responder is corrupted, and there is no recorded key, record $\mu \leftarrow \tilde{\mu}$ as the adversary. Send message ("Key", `SID`, $\bar{P}$, μ) to P.

Figure 3: The Universal Key Exchange Functionality

while executing the TLS protocol (captured by the fact that peer identities are given by the functionality and not in the setup). This is an essential difference of TLS to related protocols (e.g. SSH) where the players have already negotiated their public keys before the protocol start.

Second, the functionality defines a hybrid notion of authenticated key exchange. When the initiator is parameterized with an identity, i.e. $ID_I{=}I$, the functionality assures mutual authentication between the initiator and server. Then the functionality randomly fixes the (master) key unless a party is corrupt. On the other hand, when the initiator is invoked with an anonymous identity, i.e. $ID_I{=}\perp$, the functionality guarantees a matching conversation between the responder and some party whose identity is unknown. Consequently, the adversary can impersonate the initiator and fix the master key.[2] The corresponding case in the real world is that the environment instructs the adversary to replay the key exchange protocol with the exception that it contributes to the premaster key. The initiator is unable to terminate the session while the responder accepts the session. Technically, the functionality deploys the session identifier `SID` to determine the anonymous player. Such technicality is only feasible for a two party functionality. Recall that the `SID`s of all Turing machines in a protocol instance must be identical in the UC framework. Any player participating in the same session who is not a responder must be a potential initiator.

Third, the functionality is defined for non-adaptively corrupting adversaries and therefore excludes (perfect) forward secrecy. In fact, this exclusion makes it possible to define a universal key exchange functionality which covers both, key transport and Diffie-Hellman key agreement.

Remark. To distinguish between the authentication modes, we denote the key exchange functionality where the initiator's identity is kept secret and the initiator's identity disclosed by $\mathcal{F}_{\text{KE}}^{1}$ and $\mathcal{F}_{\text{KE}}^{2}$, respectively. The functionality $\mathcal{F}_{\text{KE}}^{(1,2)}$ qualifies a hybrid formulation thereof.

[2] Note that in case of Diffie-Hellman the key exchange functionality does not consider key control issues (see [27]). However, this has no impact on the security of secure communication sessions because the impersonator learns the master key and thus derives the session keys for the protection of the messages.

5.2 Subroutine DHE

The subroutine DHE, described in Fig. 7, is a 2-way Diffie-Hellman key exchange, whereby exponents are randomly chosen and the responder authenticates via a signature; the verification key is certified by a trusted third party and conveyed with the responder's identity. We capture the fact that the players own certified keys by exploiting the presence of an ideal certification functionality $\mathcal{F}_{\text{CERT}}$ [6], see also Fig 8 in Appendix B.1 which permits the owner of the instance to receive a signature on arbitrary messages while any player can verify the signature. The JUC-theorem ensures that multiple sessions of protocol DHE have access to the same instance $\mathcal{F}_{\text{CERT}}$ (i.e. verify arbitrary messages signed with the same signature key). Any CMA-secure signature scheme can be employed for signing (this follows from the results in [6]), and our proof (in Appendix B.1) for the UC-realization of $\mathcal{F}^1_{\text{KE}}$ by the subroutine DHE (Lemma 5) is therefore independent of the model in which the security of the signature scheme is guaranteed (e.g. in the Random Oracle model in the case of RSA-PSS).

Lemma 5 *Subroutine* DHE *in the* $\mathcal{F}_{\text{CERT}}$*-hybrid model securely realizes* $\mathcal{F}^1_{\text{KE}}$.

5.3 Subroutine DHS

The subroutine DHS, described in Fig. 9, is identical to DHE with the exception that the responder's DH exponent is certified by a trusted third party and carried within its identity. We capture the difference by reformulating DHE in the $\mathcal{F}_{\text{CA}}$-hybrid model. Functionality $\mathcal{F}_{\text{CA}}$ [6], see also Fig. 10 in Appendix B.2, serves as a trusted registration authority where the responder escrows a static DH exponent g^x and group $\langle g \rangle$ of order q in $\mathbb{Z}_p^*$. $\mathcal{F}_{\text{CA}}$ outputs the values to arbitrary players when it is invoked with the identity of the registered owner. Essentially, $\mathcal{F}_{\text{CA}}$ binds the deposit to a particular identity and captures the setup that a CA has certified static DH parameters. The JUC theorem guarantees that different sessions of DHE have access to the same instance of $\mathcal{F}_{\text{CA}}$. The UC-realization of $\mathcal{F}^1_{\text{KE}}$ by the subroutine DHS is captured with Lemma 6, proven in Appendix B.2.

Lemma 6 *Subroutine* DHS *in the* $\mathcal{F}_{\text{CA}}$*-hybrid model securely realizes* $\mathcal{F}^1_{\text{KE}}$.

5.4 Subroutine EKT

The subroutine EKT, illustrated in Fig. 11, is different in nature from previous subroutines in that the initiator transports the premaster secret encrypted with the responder's public key. We formulate EKT in the $\mathcal{F}_{\text{CPKE}}$-hybrid model that provides the players with certified public key functionality. The functionality $\mathcal{F}_{\text{CPKE}}$ [11], see also Fig. 12 in Appendix B.3, maintains a repository where any invoking player deposits plaintexts which can be accessed only by the owner of the instance. In the presence of non-adaptive[3] adversaries this functionality can be realized by any CCA2-secure encryption scheme (this follows from the results in [11]). Therefore, any CCA2-secure encryption scheme can be employed for key transport, and our proof (in Appendix B.3) for the UC-realization of $\mathcal{F}^1_{\text{KE}}$ by the subroutine DHE (Lemma 7) is therefore independent of the model in which the security of the encryption scheme is guaranteed (e.g. in the Random Oracle model

[3] Obviously, realizing $\mathcal{F}_{\text{CPKE}}$ in the presence of adaptive adversaries so that secrecy of multiple messages is preserved under the condition that the adversary has corrupted the decryptor and gained access to the secret key is a considerably stronger requirement and demands for additional techniques, such as forward secure or non-committing encryption [17, 9].

in the case of RSA-OAEP which is the recommended in the π specification). The JUC-theorem guarantees that the same instance of $\mathcal{F}_{\text{CPKE}}$ is used to encrypt multiple messages by multiple parties to a single recipient.

Lemma 7 *Subroutine* EKT *in the* $\mathcal{F}_{\text{CPKE}}$*-hybrid model securely realizes* $\mathcal{F}^1_{\text{KE}}$.

5.5 On Realizing Mutual Authentication

The framework enables the responder to opt for initiator authentication. Then, the initiator proves its identity by signing the transcript of incoming and outgoing messages whereby the verification key is certified by a trusted third party and appended to the signature. Employing the composition theorem, we capture the model by reformulating the subroutines in the $\mathcal{F}_{\text{CERT}}$-hybrid model, assuming the registration of the initiator as owner of the instance. The subroutines are extended in the following way:

Before the initiator submits the response message, it stores the message transcript in value `trscrpt`. Next, it feeds $\mathcal{F}_{\text{CERT}}$ with message ("sign", $\mathtt{SID}_{\text{CERT1}}$, `trscrpt`) where $\mathtt{SID}_{\text{CERT1}}$=($I$, $\mathtt{SID} \circ 1$) includes its identity and waits for the answer ("Signature", $\mathtt{SID}_{\text{CERT1}}$, `trscrpt`, ς). Then, the initiator places the signature ς and its identity I to the response message. When the responder receives the message, it first computes its own `trscrpt` and checks that ς is a valid signature by calling $\mathcal{F}_{\text{CERT}}$ on input ("verify", $\mathtt{SID}_{\text{CERT1}}$, `trscrpt`, ς). If the verification fails, it aborts. Otherwise, the responder continues to process the subroutine in the normal way. If the subroutine terminates, then the responder generates local output ("Key", `SID`, I, k_m).

Theorem 8 *Subroutines* DHE *in the (*$\mathcal{F}_{\text{CERT}}$*,*$\mathcal{F}_{\text{CERT}}$*)-hybrid model,* DHS *in the (*$\mathcal{F}_{\text{CA}}$*,* $\mathcal{F}_{\text{CERT}}$*)-hybrid model and* EKT *in the (*$\mathcal{F}_{\text{CPKE}}$*,*$\mathcal{F}_{\text{CERT}}$*)-hybrid model securely realizes* $\mathcal{F}^2_{\text{KE}}$.

Proof. The proof follows from the composition theorem. Lemma 5, 6, and 7 imply that the subroutines DHE, DHS and EKT UC-realize $\mathcal{F}^1_{\text{KE}}$. It remains to show that $\mathcal{F}^1_{\text{KE}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model securely realizes $\mathcal{F}^2_{\text{KE}}$. It is easy to see that $\mathcal{Z}$'s distribution when it interacts with the dummy players calling functionality $\mathcal{F}^1_{\text{KE}}$ is identical to the its distribution when it communicates with the dummy players invoking $\mathcal{F}^2_{\text{KE}}$ except that the responder's output includes the initiator identity. However, by calling $\mathcal{F}_{\text{CERT}}$ the initiator commits to its identity. Hence, the output distributions of $\mathcal{F}^1_{\text{KE}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid are indistinguishable from an emulation of $\mathcal{F}^2_{\text{KE}}$. ■

6 TLS UC-Realizes Secure Communication Sessions

The natural abstraction of TLS is to allow secure communication between players in a single protocol instance. While the handshake protocol aims at securely sharing uniformly distributed session keys, the record layer protocol provides authenticated encryption of session messages.

6.1 Universal Secure Communication Sessions

Secure communication sessions have been discussed in [5, 13] for the general case in which all players are authenticated. We refine the functionality and relax the requirements to the universal model of authentication in the post-specified setting, where a player learns the identity of its peer during the execution of the protocol and must cope with impersonation attacks against the initiator, provided the environment keeps the initiator's identity secret. In which case, we have to expect a real-world adversary that plays the role of the initiator by intercepting the first two protocol rounds, choosing own premaster secret, and completing the protocol in the normal way. The initiator will be unable to terminate the session. Nevertheless, the responder accepts the session and answers to the adversary, mimicking arbitrary party. We capture the requirements by formulating a universal secure communication sessions functionality $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$ in Fig. 4. Let us highlight some characteristics of $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$ in the following:

Functionality $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$

$\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$ proceeds as follows, when parameterized by a leakage function $l : \{0,1\}^* \rightarrow \{0,1\}^*$.

- Upon receiving an input ("establish-session", `SID`, ID_I) from some party, where $ID_I \in (\perp, I)$, record ID_I as initiator, and send the message to the adversary. Upon receiving input ("establish-session", `SID`, R) from some party, record R as responder, and forward the message to the adversary.
- Upon receiving a value ("impersonate", `SID`) from the adversary, do:If ($ID_I{=}\perp$), check that no `ready` entry exists, and record the adversary as initiator. Else ignore the message.
- Upon receiving a value ("send", `SID`, m, $\bar{P}$) from party P, which is either initiator or responder, check that a record (`SID`, P, $\bar{P}$) exists, record `ready` (if there is no such entry) and send ("sent", `SID`, $l(m)$) to the adversary and a private delayed value ("receive", `SID`, m, P) to $\bar{P}$. Else ignore the message. If the sender is corrupted, then disclose m to the adversary. Next, if the adversary provides m' and no output has been written to the receiver, then send ("send", `SID`, m', P') to the receiver unless P' is an identity of an uncorrupted party.

Figure 4: The Universal Secure Communication Sessions Functionality

First, the functionality handles a uni- and bi-directional model of authentication (as in the universal key exchange functionality). The latter is accomplished by invoking the players with their own identity. The first is realized by invoking the initiator with an empty identity value $\perp$ allowing the adversary to mount an impersonation attack. The functionality proceeds in the usual way except that a secure session is established between the adversary and the responder.

Second, the functionality guarantees that the adversary gains no information other than some side channel information about the transmitted plaintext m, expressed via a leakage function $l(m)$, when the adversary has neither impersonated nor corrupted a player. In particular, the information leakage includes the length of m and some information concerning the transmitted messages' source and destination; thus, modeling network information about the TLS-protected channel from lower-layer protocols and higher-layer protocols prior to their processing by the record layer. Further, the leakage reveals the error messages provided by the TLS alert protocol, when either party fails to complete the protocol.

Third, the session identifier `SID` assures that the functionality may address the initiator even though its identity is undisclosed (because it knows the responder's identity and the underlying system model permits a party, i.e. the initiator, to interact with the functionality with an identical session identifier). This is so because TLS runs above transport-layer protocols which provide the players with routing information (e.g. IP address, domain) and a establish a channel for the communication session. Furthermore, these protocols typically ensure that the channel is locally fresh by exchanging a pair of nonces. The environment mimics the task of these surrounding processes by activating the players with session identifier $\mathtt{SID}$.

Fourth, the functionality does not pre-process the messages to be sent to the peer. In fact, we assume that the environment "prepares" the messages, i.e. compresses, fragments and adds a sequence number into the encoding. Otherwise, the functionality must provide the technicalities. This would unnecessarily complicate the formulation of the ideal functionality.

Lastly, the functionality manages an internal ready state. This technicality ensures that in the responder-only model of authentication the adversary cannot impersonate the initiator after the responder agreed upon the session keys and switched into the pending state waiting for the transmission.

6.2 Protocol ρ realizes $\mathcal{F}_{\text{SCS}}^{(1,2)}$

In Fig. 5 we apply Theorem 8 and reformulate protocol ρ in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model. The general Universal Composability theorem guarantees that no probabilistic polynomial time-bounded environment distinguishes between the case that it observes an instance of TLS executing the subroutines DHE, DHS and EKT and the case that it interacts with a TLS instance where the subroutines are replaced by the ideal key exchange functionality. We are now ready to state our main theorem.

Theorem 9 *Protocol ρ in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model securely realizes $\mathcal{F}_{\text{SCS}}^{(1,2)}$.*

Proof. Let $\mathcal{A}$ be a real-world adversary that operates against ρ. We construct an ideal-world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between the case that it interacts with $\mathcal{A}$ and parties running ρ in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model or with $\mathcal{S}$ in the ideal world for $\mathcal{F}_{\text{SCS}}^{(1,2)}$. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and mimics an interaction with players executing ρ. It tries to make the internal protocol simulation consistent with the real protocol execution and the limitation that it has no information about the transmitted message m other than its length $l(m)$. The simulator allows the adversary $\mathcal{A}$ to attack the simulated protocol execution in arbitrary way throughout the simulation. $\mathcal{S}$ emulates the protocol execution in such a way that $\mathcal{A}$ thinks that it intercepts a real-world execution of ρ, and such that its interaction with $\mathcal{Z}$ is distributed computationally indistinguishable from that observed by the environment in the real-world execution.

In detail, the simulator proceeds in the following way:

1. **Simulating invocation of I.** Upon receiving ("establish-session", $\mathtt{SID}$, ID_I) from $\mathcal{F}_{\text{SCS}}^{(1,2)}$, $\mathcal{S}$ feeds $\mathcal{A}$ with the init message (r_I) where $r_I \overset{r}{\leftarrow} \{0,1\}^{p_1(k)}$.

2. **Simulating invocation of R.** Upon receiving ("establish-session", $\mathtt{SID}$, R) from $\mathcal{F}_{\text{SCS}}^{(1,2)}$, $\mathcal{S}$ waits for receipt of an init message (r'_I) from $\mathcal{A}$. Then, it chooses a nonce $r_R \overset{r}{\leftarrow} \{0,1\}^{p_2(k)}$ and feeds $\mathcal{A}$ with the response message (r_R, R). Finally, it calls $\mathcal{F}_{\text{KE}}^{(1,2)}$ on query ("establish-key", $\mathtt{SID}'_{\text{KE}}$, R), where $\mathtt{SID}'_{\text{KE}}$=($\mathtt{SID} \circ \mathtt{r}'_{\mathtt{I}}|\mathtt{r}_{\mathtt{R}}$).

3. **Simulating receipt of a response message by I.** Upon $\mathcal{A}$ delivers the message (r'_R, P') to I, $\mathcal{S}$ proceeds as follows:

   (a) $\mathcal{S}$ verifies that I has previously sent the init message (r_I).

   (b) $\mathcal{S}$ checks that P'=R. Otherwise, it aborts the simulation.

Protocol ρ

(a) Upon activation with query (“establish-session”, $\mathtt{SID}$, ID_I) by $\mathcal{Z}$, where $ID_I \in (\perp, I)$, the initiator sends the init message (r_I) where $r_I \xleftarrow{r} \{0,1\}^{p_1(k)}$ is a nonce. Upon activation with query (“establish-key”, $\mathtt{SID}$, R) by $\mathcal{Z}$, the responder waits for the receipt of the init message. It responds with own nonce $r_R \xleftarrow{r} \{0,1\}^{p_2(k)}$ and initializes a copy of $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ with session identifier $\mathtt{SID}_{\mathrm{KE}}{=}(r_I|r_R)$ by sending query (“establish-key”, $\mathtt{SID}_{\mathrm{KE}}$, R) to $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$.

(b) Upon receiving the response message, the initiator calls $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$ with session identifier $\mathtt{SID}_{\mathrm{KE}}{=}(r_I|r_R)$ on query (“establish-key”, $\mathtt{SID}_{\mathrm{KE}}$, ID_I) and waits for the delivery of output (“Key”, $\mathtt{SID}_{\mathrm{KE}}$, R, μ). It then computes the session keys $(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_\mu(l_2)$ and the finished value $F_I \leftarrow \mathtt{PRF}_\mu(l_3)$. Additionally, the initiator sends the final initiator message $(\mathtt{E}_{k_e^I}(F_I|\mathtt{HMAC}_{k_a^I}(F_I)))$.

(c) When the responder receives the final initiator message (α), it first waits for the delivery of (“Key”, $\mathtt{SID}_{\mathrm{KE}}$, ID_I, μ) from $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$. Then, the responder computes in the same way the session keys $(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_\mu(l_2)$ for the players. It decrypts the final initiator message $(F_I|t_I) \leftarrow \mathtt{D}_{k_e^I}(\alpha)$ and verifies that $F_I \leftarrow \mathtt{PRF}_\mu(l_3)$ and $t_I \leftarrow \mathtt{HMAC}_{k_a^I}(F_I)$. If the verification fails, it aborts. Otherwise, it computes the finished value $F_R \leftarrow \mathtt{PRF}_\mu(l_4)$ and sends the final responder message $(\mathtt{E}_{k_e^R}(F_R|\mathtt{HMAC}_{k_a^R}(F_R)))$.

(d) Upon delivery of the final responder message (β), the initiator decrypts the message $(F_R|t_R) \leftarrow \mathtt{D}_{k_e^R}(\beta)$. Then, it verifies that $F_R \leftarrow \mathtt{PRF}_\mu(l_4)$ and $t_R \leftarrow \mathtt{HMAC}_{k_a^R}(F_R)$. If the verification fails, it aborts.

(e) Once the session keys are agreed upon, the sender $P \in (I, R)$ waits for the transmission notification (“send”, $\mathtt{SID}$, m, $\bar{P}$) from $\mathcal{Z}$. It then sends $\mathtt{E}_{k_e^P}(m|t_m)$ whereby message m is authenticated through the tag $t_m \leftarrow \mathtt{HMAC}_{k_a^P}(m)$. Upon receiving the message γ, the receiver $\bar{P}$ decrypts the message $(m|t_m) \leftarrow \mathtt{D}_{k_e^P}(\gamma)$ and verifies that $t_m \leftarrow \mathtt{HMAC}_{k_a^P}(m)$. If the verification fails, it aborts. Otherwise, the receiver accepts the message and makes the local output (“receive”, $\mathtt{SID}$, m, P) to $\mathcal{Z}$.

Figure 5: The full TLS Framework Structure, in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid Model

(c) $\mathcal{S}$ mimics on behalf of I the master key generation by invoking a copy of $\mathcal{F}_{\text{KE}}^{(1,2)}$. The master key is obtained by handing $\mathcal{F}_{\text{KE}}^{(1,2)}$ the message ("establish-key", $\texttt{SID}_{\text{KE}}$, ID_I), where $\texttt{SID}_{\text{KE}}$=($\texttt{SID} \circ \texttt{r}_\texttt{I}|\texttt{r}'_\texttt{R}$) and waiting for the delivery of the response message ("Key", $\texttt{SID}_{\text{KE}}$, R, μ). Otherwise, $\mathcal{S}$ terminates with an internal error message (because there was no matching activation of the same instance of $\mathcal{F}_{\text{KE}}^{(1,2)}$ in form of a query ("establish-key", $\texttt{SID}_{\text{KE}}$, R) by the simulator on behalf of the responder).

(d) $\mathcal{S}$ defines the master key μ, the session keys $(k_e^I, k_a^I, k_e^R, k_a^R)$, and the finished value F_I to be random values Δ_{k_m}, $(\Delta_{k_e^I}, \Delta_{k_a^I}, \Delta_{k_e^R}, \Delta_{k_a^R})$, and Δ_{F_I} chosen from the appropriate spaces, respectively.

(e) $\mathcal{S}$ feeds $\mathcal{A}$ with the final initiator message $(\texttt{E}_{\Delta_{k_e^I}}(\Delta_{F_I}|t_I))$, where $t_I \leftarrow \texttt{HMAC}_{\Delta_{k_a^I}}(\Delta_{F_I})$.

4. **Simulating receipt of a final initiator message by R.** When $\mathcal{A}$ delivers the message (α) to R, $\mathcal{S}$ proceeds as follows:

(a) $\mathcal{S}$ verifies that it has previously received an init message (r'_I) and sent a response message (r_R, R).

(b) $\mathcal{S}$ waits for the master key by mimicking the key establishment process of $\mathcal{F}_{\text{KE}}^{(1,2)}$. Now we distinguish between the following two distinct cases.

Case 1 (no impersonation): If $\mathcal{S}$ receives an answer ("Key", $\texttt{SID}_{\text{KE}}$, ID_I, μ) from $\mathcal{F}_{\text{KE}}^{(1,2)}$, then no impersonation attack has occurred. In this case $\mathcal{S}$ uses for k_m, $(k_e^I, k_a^I, k_e^R, k_a^R)$, and F_I exactly the same values Δ_{k_m}, $(\Delta_{k_e^I}, \Delta_{k_a^I}, \Delta_{k_e^R}, \Delta_{k_a^R})$, and Δ_{F_I} that it has chosen on behalf of the initiator before. Then, it waits for the delivery of the final initiator message and applies the session keys to decrypt $(F'_I|t'_I) \leftarrow \texttt{D}_{\Delta_{k_e^I}}(\alpha)$. $\mathcal{S}$ compares whether $F'_I = \Delta_{F_I}$ and $t'_I = t_I$. If the verification fails, it aborts the simulation. Otherwise, it chooses F_R to be a random value Δ_{F_R} from the same space and feeds $\mathcal{A}$ with the final responder message $(\texttt{E}_{\Delta_{k_e^R}}(\Delta_{F_R}|t_R))$ where $t_R \leftarrow \texttt{HMAC}_{\Delta_{k_a^R}}(\Delta_{F_R})$. Then, $\mathcal{S}$ prepares for the secure message exchange on behalf of R.

Case 2 (impersonation): If $\mathcal{S}$ receives an answer ("Key", $\texttt{SID}'_{\text{KE}}$, P', $\tilde{\mu}$), then the original master key has been modified by the adversary implying the impersonation attack framing the initiator. In this case $\mathcal{S}$ computes $(k_e^I, k_a^I, k_e^R, k_a^R)$, and F_I as specified in the protocol, i.e. $(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \texttt{PRF}_{\tilde{\mu}}(l_2)$, and $F_I \leftarrow \texttt{PRF}_{\tilde{\mu}}(l_3)$. Then, it waits for the delivery of the final initiator message and decrypts $(F'_I|t'_I) \leftarrow \texttt{D}_{k_e^I}(\alpha)$. $\mathcal{S}$ compares whether $F'_I = F_I$ and $t'_I = \texttt{HMAC}_{k_a^I}(F_I)$. If the verification fails, it aborts the simulation. Otherwise, $\mathcal{S}$ computes $F_R \leftarrow \texttt{PRF}_{\tilde{\mu}}(l_4)$, and feeds $\mathcal{A}$ with the final responder message $(\texttt{E}_{k_e^R}(F_R|\texttt{HMAC}_{k_a^R}(F_R)))$. Finally, $\mathcal{S}$ sends ("impersonate", $\texttt{SID}$) to $\mathcal{F}_{\text{SCS}}^{(1,2)}$. This is exactly the point in the simulation where the adversary has impersonated the unauthenticated party. Then, $\mathcal{S}$ continues the simulation with the exception that the interaction proceeds with $\mathcal{A}$ and I aborts the protocol.

Note that in all subsequent simulation steps, $\mathcal{S}$ uses session keys (k_e^P, k_a^P) for $P \in (I, R)$ and finished values F_I and F_R obtained from one of the above two cases.

5. **Simulating receipt of a final responder message by I.** When $\mathcal{A}$ delivers the message (β) to an uncorrupted I, $\mathcal{S}$ proceeds as follows:

(a) $\mathcal{S}$ verifies that it has previously sent an init message (r_I), received a response message (r'_R, P'), and sent a final initiator message ($\mathtt{E}_{\Delta_{k_e^I}}(\Delta_{F_I}|t_I)$).

(b) $\mathcal{S}$ uses its own session keys ($\Delta_{k_e^R}, \Delta_{k_a^R}$) to decrypt β obtaining $F'_R|t'_R$. Since no responder impersonation attacks may occur it aborts the simulation if $F'_R \neq \Delta_{F_R}$ or $t'_R \neq t_R$ whereby Δ_{F_R} and t_R are the values used by $\mathcal{S}$ on behalf of R in the previous simulation step 4b (case 1). If the simulation does not abort then $\mathcal{S}$ prepares for the secure message exchange on behalf of I.

6. **Simulating Message Transmission.** Upon receiving ("sent", $\mathtt{SID}$, $l(m)$) from $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$, $\mathcal{S}$ extracts from $l(m)$ the sender and receiver identities. It then chooses a random message $\Delta_m \xleftarrow{r} \{0,1\}^{l(m)}$ and feeds $\mathcal{A}$ with message $\mathtt{E}_{k_e^P}(\Delta_m|t_{\Delta_m})$ where $t_{\Delta_m} \leftarrow \mathtt{HMAC}_{k_a^P}(\Delta_m)$.

7. **Simulating Message Reception.** Upon receiving the message (γ), the receiver decrypts the message ($\Delta'_{m'}$,$t'_{\Delta_{m'}}$) $\leftarrow \mathtt{D}_{k_e^P}(\gamma)$ and then verifies that $t'_{\Delta_{m'}} \leftarrow \mathtt{HMAC}_{k_a^P}(\Delta_m)$ using its own keys. If the verification fails, it aborts. Otherwise, $\mathcal{S}$ signals $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$ to send the message.

8. **Simulating Static Corruption** If one of the parties gets corrupted, then $\mathcal{S}$ proceeds by emulating a ρ protocol session, just as a honest party would play it. In particular, $\mathcal{S}$ uses the message m transmitted by $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$ in the emulation of the last protocol round.

We now prove that the simulator $\mathcal{S}$ is such that no environment $\mathcal{Z}$ can distinguish between the ideal execution of $\mathcal{F}_{\mathrm{SCS}}^{(1,2)}$ and $\mathcal{S}$, and the real execution of ρ and $\mathcal{A}$ in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid model. To prove that the environment's view is indistinguishable in the two worlds, we define a sequence of hybrid distributions, $\boldsymbol{\mathcal{H}_1}$ and $\boldsymbol{\mathcal{H}_2}$, where we make some modifications, starting from the real-world protocol execution and ending at the ideal-world execution. We show

$$\mathtt{UC{-}EXEC}^{\mathcal{F}_{\mathrm{KE}}^{(1,2)}}_{\rho,\mathcal{A},\mathcal{Z}} \approx \boldsymbol{\mathcal{H}_1} \approx \boldsymbol{\mathcal{H}_2} \approx \mathtt{UC{-}EXEC}_{\mathcal{F}_{\mathrm{SCS}}^{(1,2)},\mathcal{S},\mathcal{Z}} \tag{1}$$

The distributions are defined as follows:

- $\boldsymbol{\mathcal{H}_1}$ takes the distribution of the output of $\mathcal{Z}$ which is identical to a real execution of $\mathcal{A}$ running ρ in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid model with the following exception. When the protocol ρ instructs the initiator (respectively the responder) to evaluate the pseudo-random function $\mathtt{PRF}()$ with some random key Δ_{k_m} in the range of the master secret in order to compute the session keys (k_e^P, k_a^P) for $P \in (I, R)$ and the finished values F_I and F_R, we replace the outputs with the independently chosen random values ($\Delta_{k_e^P}, \Delta_{k_a^P}$), Δ_{F_I}, and Δ_{F_R}, respectively, all in the range of $\mathtt{PRF}()$.

- $\boldsymbol{\mathcal{H}_2}$ is identical to $\boldsymbol{\mathcal{H}_1}$ with the following exception. The simulation fails if on behalf of the responder (resp. initiator) it receives a message Δ_m which it decrypts with k_e^P and obtains a valid authentication tag which has not been generated by the simulator on behalf of the responder (resp. initiator) before. Since the authentication tags are generated via the $\mathtt{HMAC}()$ function the simulation failure occurs whenever the adversary forges a corresponding tag.

Claim 10 *If* $\mathtt{PRF}()$ *is a secure pseudo-random function, then* $\mathtt{UC{-}EXEC}^{\mathcal{F}_{\mathrm{KE}}^{(1,2)}}_{\rho,\mathcal{A},\mathcal{Z}} \approx \boldsymbol{\mathcal{H}_1}$.

Proof. Assume that such environment $\mathcal{Z}$ that distinguishes with non-negligible simulation slice between the interaction $\mathtt{UC-EXEC}^{\mathcal{F}_{\mathrm{KE}}^{(1,2)}}_{\rho,\mathcal{A},\mathcal{Z}}$ and $\mathcal{H}_{\mathbf{1}}$ exists. We construct an adversary $\mathcal{D}$ that has access to the black-box oracle $\mathcal{O}_{\mathcal{P}}$. The oracle computes throughout the whole simulation either $\mathtt{PRF}_{\Delta_{k_m}}()$ for some randomly chosen secret Δ_{k_m} or a truly random function with the same range.

$\mathcal{D}$ runs a copy of $\mathcal{Z}$ and mimics the roles of $\mathcal{A}$ and the parties. It emulates the interaction with parties running ρ in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid model until it is required to compute the session keys (k_e^P, k_a^P) or the finished values F_I or F_R which it obtains through the corresponding call to $\mathcal{O}_{\mathcal{P}}$. $\mathcal{D}$ interpolates between $\mathtt{UC-EXEC}^{\mathcal{F}_{\mathrm{KE}}^{(1,2)}}_{\rho,\mathcal{A},\mathcal{Z}}$ (this is the case where $\mathcal{O}_{\mathcal{P}}$ contains $\mathtt{PRF}_{\Delta_{k_m}}()$) and $\mathcal{H}_{\mathbf{1}}$ (this is the case where $\mathcal{O}_{\mathcal{P}}$ contains the truly random function) so that based on the output of $\mathcal{O}_{\mathcal{P}}$, $\mathcal{D}$ can distinguish with non-negligible probability between $\mathtt{UC-EXEC}^{\mathcal{F}_{\mathrm{KE}}^{(1,2)}}_{\rho,\mathcal{A},\mathcal{Z}}$ and $\mathcal{H}_{\mathbf{a}}$ for any $\mathcal{Z}$. □

Claim 11 *If* $\mathtt{HMAC}()$ *is a WUF-CMA secure message authentication function, then* $\mathcal{H}_{\mathbf{1}} \approx \mathcal{H}_{\mathbf{2}}$.

Proof.Assume that there exists an environment $\mathcal{Z}$ that distinguishes with non-negligible simulation slice between the interaction $\mathcal{H}_{\mathbf{1}}$ and $\mathcal{H}_{\mathbf{2}}$. We specify an adversary $\mathcal{F}$ that has access to an authentication oracle $\mathcal{O}_{\mathcal{M}}$ which on input some message m outputs the corresponding authentication tag $\mathtt{HMAC}_{\Delta_{k_a^P}}(m)$ using some initially fixed random key $\Delta_{k_a^P}$ unknown to $\mathcal{F}$ and the corresponding verification oracle $\mathcal{V}_{\mathcal{M}}$ that on input a message m and a candidate authentication tag t outputs $\mathtt{valid}$ if $t \leftarrow \mathtt{HMAC}_{\Delta_{k_a^P}}(m)$, otherwise it returns $\mathtt{invalid}$. We show how $\mathcal{F}$ can use $\mathcal{Z}$ in order to output a valid tag t' for some message m' which has not been previously queried to $\mathcal{O}_{\mathcal{M}}$. In fact, $\mathcal{F}$ emulates the execution of ρ in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid model and mimics for $\mathcal{Z}$ the role of the adversary and the players with the exception that the required authentication tags for the computed protocol messages $\mathcal{F}$ obtains from $\mathcal{O}_{\mathcal{M}}$. Thus, the environment can easily recognize when some forgery is received during the emulation of the protocol with respect to the adversary. This forgery is also the output of $\mathcal{F}$. Thus, unless the simulation fails both hybrids $\mathcal{H}_{\mathbf{1}}$ and $\mathcal{H}_{\mathbf{2}}$ proceed identical. By assumption the simulation fails only with negligible probability. □

Claim 12 *If* $\mathtt{E}(), \mathtt{D}()$ *is an IND-CPA secure encryption scheme, then* $\mathcal{H}_{\mathbf{2}} \approx \mathtt{UC-EXEC}_{\mathcal{F}_{\mathrm{SCS}}^{(1,2)},\mathcal{S},\mathcal{Z}}$.

Proof. Assume that such environment $\mathcal{Z}$ that distinguishes with non-negligible simulation slice between the interaction $\mathcal{H}_{\mathbf{2}}$ and $\mathtt{UC-EXEC}_{\mathcal{F}_{\mathrm{SCS}}^{(1,2)},\mathcal{S},\mathcal{Z}}$ exists. We construct an adversary $\mathcal{D}$ that has access to the *Real-Or-Random* encryption oracle $\mathcal{O}_{\mathcal{E}}$ that on input a message m outputs $\mathtt{E}_{\Delta_{k_e^P}}(m_b)$ using some initially fixed random unknown key $\Delta_{k_e^P}$ and bit b such that in case $b = 0$ the oracle uses $m_0 \leftarrow m$ (this is the message from the real world) and in case $b = 1$ it uses $m_1 \leftarrow \tilde{m}$ for some $\tilde{m} \neq m$ (this is the message chosen by the simulator). We show how $\mathcal{S}_{\mathcal{E}}$ breaks the security of the symmetric encryption scheme, i.e. decides on b with a probability non-negligibly larger than 1/2.

$\mathcal{D}$ runs a copy of $\mathcal{A}$ and emulates the interaction with parties running ρ in the $\mathcal{F}_{\mathrm{KE}}^{(1,2)}$-hybrid model except that whenever $\mathcal{D}$ is required to obtain a ciphertext on some message m' (which can be one of the finished messages $F_I|t_{F_I}$ or $F_R|t_{F_R}$, or the securely transmitted message $m|t_m$) it queries $\mathcal{O}_{\mathcal{E}}$ on m'. In this way $\mathcal{D}$ interpolates between $\mathcal{H}_{\mathbf{2}}$ (this is the case where the oracle encrypts a real message, i.e. $b = 0$) and $\mathtt{UC-EXEC}_{\mathcal{F}_{\mathrm{SCS}}^{(1,2)},\mathcal{S},\mathcal{Z}}$ (this is the case where the oracle encrypts a different message, i.e. $b = 1$). Based on the output of $\mathcal{Z}$ $\mathcal{D}$ can decide on the bit b used by $\mathcal{O}_\mathcal{E}$. Thus, by assumption $\mathcal{H}_\mathbf{2} \approx \texttt{UC}-\texttt{EXEC}_{\mathcal{F}_{\text{SCS}}^{(1,2)},\mathcal{S},\mathcal{Z}}$ for any $\mathcal{Z}$. □

This completes the simulation and the proof of the theorem. ■

7 Conclusion

We have analyzed the TLS protocol family in the framework of Universal Composition. We have shown that the complete TLS protocol framework securely realizes secure communication sessions. Thus, future analysis of composite protocols can be considerably simplified by calling the secure communication functionality in the hybrid-model reformulation. The composition theorem preserves that security holds under general composition with arbitrary players. Furthermore, since the ideal functionality is free from any probabilism, it may be expressed in an abstract term algebra, enabling Dolev-Yao style proofs. It allows applying automated proof tools while preserving soundness and composition. In many analysis of Internet protocols TLS has been abstracted away in a Dolev-Yao-style and assumed to provide secure communication channels [25, 23, 24]. Our analysis is a first step towards showing whether this claim is founded.

Acknowledgment

We would like to thank Dennis Hofheinz, Aggelos Kiayias, Ralf Küsters, and Ivan Visconti for fruitful discussions and their valuable feedback.

References

[1] B. Aboba, L. J. Blunk, J. R. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible authentication protocol (EAP). Internet RFC 3748, June 2004.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO*, pages 1–15, 1996.

[3] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[4] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[6] R. Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–233. IEEE Computer Society, 2004. Full version available on `http://eprint.iacr.org/2003/239`.

[7] R. Canetti. Security and composition of cryptographic protocols: A tutorial. Cryptology ePrint Archive, Report 2006/465, 2006. `http://eprint.iacr.org/`.

[8] R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[9] R. Canetti, S. Halevi, and J. Katz. Adaptively-secure, non-interactive public-key encryption. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 150–168. Springer, 2005.

[10] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, 2005.

[11] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *TCC*, pages 380–403, 2006.

[12] R. Canetti and H. Krawczyk. Security analysis of ike's signature-based key-exchange protocol. In Yung [42], pages 143–161.

[13] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, pages 337–351, 2002.

[14] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing chosen-ciphertext security. Cryptology ePrint Archive, Report 2003/174, 2003. `http://eprint.iacr.org/`.

[15] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[16] R. Canetti and T. Rabin. Universal composition with joint state. In *CRYPTO*, pages 265–281, 2003.

[17] I. Damgård and J. B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In M. Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2000.

[18] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol, version 1.1. RFC 4346, IETF, 2006. Proposed Standard.

[19] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[20] P.-A. Fouque, D. Pointcheval, and S. Zimmer. HMAC is a randomness extractor and applications to TLS. In *AsiaCCS '08*, pages 21–32. ACM Press, 2008.

[21] C. Gentry, P. D. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2006.

[22] T. Groß. Security analysis of the SAML single sign-on browser/artifact profile. In *Annual Computer Security Applications Conference*. IEEE Computer Society, 2003.

[23] T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Browser model for security analysis of browser-based protocols. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 489–508. Springer, 2005.

[24] T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Proving a ws-federation passive requestor profile with a browser model. In *Workshop on Secure Web Services*. ACM Press, 2005.

[25] S. Hansen, J. Skriver, and H. Nielson. Using static analysis to validate the saml single sign-on protocol. In *Proceedings of the 2005 Workshop on Issues in the Theory of Security*, 2005.

[26] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of ieee 802.11i and tls. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 2–15. ACM, 2005.

[27] D. Hofheinz, J. Müller-Quade, and R. Steinwandt. Initiator-resilient universally composable key exchange. In E. Snekkenes and D. Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 61–84. Springer, 2003.

[28] J. Jonsson. Security proofs for the RSA-PSS signature scheme and its variants. Cryptology ePrint Archive, Report 2001/053, 2001. `http://eprint.iacr.org/`.

[29] J. Jonsson and B. Kaliski. On the security of rsa encryption in tls. In Yung [42], pages 127–142.

[30] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[31] D. Kidron and Y. Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. Cryptology ePrint Archive, Report 2007/478, 2007.

[32] D. Kormann and A. Rubin. Risks of the passport single signon protocol. *Computer Networks*, 33(1–6):51–58, 2000.

[33] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, 2000.

[34] A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In *ACISP*, pages 53–68, 2008.

[35] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of ssl 3.0. In *Proceedings of the 7th conference on USENIX Security Symposium*, pages 16–16. USENIX Association, 1998.

[36] P. Morrissey, N.P.Smart, and B. Warinschi. A modular security analysis of the tls handshake protocol. Cryptology ePrint Archive, Report 2008/236, 2008. `http://eprint.iacr.org/`.

[37] K. Ogata and K. Futatsugi. Equational approach to formal analysis of tls. In *ICDCS*, pages 795–804. IEEE Computer Society, 2005.

[38] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Computer and System Security*, 2(3):332–351, 1999.

[39] B. Pfitzmann and M. Waidner. Analysis of liberty single-signon with enabled clients. *IEEE Internet Computing*, 7(6):38–44, 2003.

[40] B. Schneier and D. Wagner. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, 1996.

[41] V. Shoup. On formal models for secure key exchange (version 4). Technical report, IBM Research Report RZ 3120, November 15 1999.

[42] M. Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.

A Impossibility of UC-secure TLS Handshakes

The main pillar of the TLS framework are the various handshake protocols that negotiate cryptographic key material for the instantiation of secure communication channels. We would like to claim that the analysis can be carried out in the spirit of a hybrid-model reformulation. We wish to show that the native TLS composition, i.e., establish the session keys by the handshake protocols and then use some cryptography to build secure channels by the record-layer protocols, is secure under the universal composable notion. That means, we would like to show that the handshake protocols securely emulate ideal key exchange and the record layer simply syncs the keys to establish the secure session as suggested in [13]. A separated consideration of the handshake protocols would have wide applicability and is of independent interest because there exist also protocols that take advantage of the functionality provided by the handshake protocol (e.g., the IEEE 802.11 EAP protocol family [1]). Unfortunately, it turns out that the handshake protocols are neither secure under the standard nor relaxed notion of UC security. This is due to a commitment problem in the confirmation of the session keys. The environment can test whether the session keys origin from the interaction with the real protocol in presence of $\mathcal{A}$ or ideal protocol in presence of $\mathcal{S}$.

A.1 Protocol π does not UC-realize $\mathcal{F}_{\text{KE+}}^{(1,2)}$

We would like to claim that the handshake protocol UC-realizes an ideal key exchange functionality. Unfortunately, such a strong claim does not hold. To understand why, we formulate the handshake protocols π in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model. Protocol π in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model is illustrated in Fig. 6. To capture the ideal-world security requirements of π, we make use of functionality $\mathcal{F}_{\text{KE+}}^{(1,2)}$. Essentially, it is a copy of $\mathcal{F}_{\text{KE}}^{(1,2)}$ with the exception that $\mathcal{F}_{\text{KE+}}^{(1,2)}$ outputs the session keys κ (instead of the master secret μ).

Claim 13 *Protocol π in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model does not securely realize $\mathcal{F}_{\text{KE+}}^{(1,2)}$.*

Proof.(Sketch) There exists an environment $\mathcal{Z}$ that distinguishes with non-negligible probability whether it communicates with the players interacting with the ideal key exchange functionality $\mathcal{F}_{\text{KE+}}^{(1,2)}$ in the presence of adversary $\mathcal{S}$ and players interacting with protocol π in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-hybrid model in front of the adversary $\mathcal{A}$. We construct such $\mathcal{Z}$ as follows:

$\mathcal{Z}$ activates first I with input ("establish-key, $\mathtt{SID}$, ID_I), instructs the adversary to deliver the init message (r_I), and records the value. Similarly, it activates R with input ("establish-key, $\mathtt{SID}$, R), instructs the adversary to deliver the response message (r_R), and records the value. It then instructs the adversary to send the final initiator message ($\mathtt{E}_{k_e^I}(F_I|\mathtt{HMAC}_{k_a^I}(F_I))$), records the message, and waits for the output ("Key", $\mathtt{SID}$, ID_I, κ) from R. Recall that $\kappa{=}(k_e^I, k_a^I, k_e^R, k_a^R)$ are the session keys. Finally, it decrypts the final initiator message ($F_I|t_I$) and verifies that $t_I \leftarrow \mathtt{HMAC}_{k_a^I}(F_I)$. If the verification fails, $\mathcal{Z}$ outputs "ideal world", otherwise it outputs "real world".

Obviously, when $\mathcal{Z}$ communicates with protocol π in the presence of adversary $\mathcal{A}$, it outputs "real world". By contrast, when interacting with the ideal key exchange functionality $\mathcal{F}_{\text{KE+}}^{(1,2)}$ in the presence of adversary $\mathcal{S}$, the session keys, say $\tilde{\kappa}{=}(\tilde{k}_e^I, \tilde{k}_a^I, \tilde{k}_e^R, \tilde{k}_a^R)$, are independently and randomly chosen by the functionality. The keys are potentially different from κ. $\mathcal{S}$ has to come up with the session keys κ without ever seeing the master secret μ. Otherwise $\mathcal{Z}$ decrypts the final initiator message using $\tilde{k}_e^I$ and notices that $t_I \not\leftarrow \mathtt{HMAC}_{\tilde{k}_a^I}(F_I)$. Consequently the probability that $\mathcal{S}$ simulates the required session keys is $1/k$ and $\mathcal{Z}$ distinguishes between the ideal and real process with overwhelming probability. ■

A.2 Protocol π does not relaxed UC-realize $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(1,2)}$

We wish to formulate a relaxed definition of the key exchange functionality $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(1,2)}$ and would like to claim that protocol π relaxed UC-realizes $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(1,2)}$. The functionality differs from the previous definition in that it sets the session key to the local output of the non-information oracle $\mathcal{N}$ unless the initiator is uncorrupted. Otherwise, it fixes the session key as the adversary. See [27, p. 117] for a definition. Unfortunately, such a claim does not hold either.

Claim 14 *Protocol π does not relaxed UC-secure realize $\mathcal{F}_{\text{KE+}}^{\mathcal{N},(1,2)}$.*

Proof.(Sketch) We use the equivalence between relaxed-UC security and SK-security, and show that π is not an SK-secure key exchange protocol, even in front of a passive adversary. Adversary $\mathcal{A}$ instructs honest parties to run a session of the π protocol, of which $\mathcal{A}$ stores all messages. Then, $\mathcal{A}$ makes a test query for that session, and receives a pair of keys $(k_e^I, k_a^I, k_e^R, k_a^R)$ that are either the keys corresponding to the executed session or random keys, with probability one half. In order to decide whether it sees real or random keys, $\mathcal{A}$ simply uses those keys to decrypt the third and fourth messages of the π session it monitored, checks the MACs, and decides that it received the real keys if the decryption and MAC verification succeeded. This adversary makes the correct guess with overwhelming probability, contradicting the SK-security definition. ■

Remark. The impossibility of simulating the key exchange functionality is due to the fact that the players commit to the finished value by tagging before encrypting the message. In fact, the problem is similar to the general commitment problem presented in [8]. Since the finished values are identically computed, the impossibility result applies to all cipher suites. However, the result does not indicate that the handshake protocols are insecure. In fact, the implication is that the desired native composition is infeasible and we have to search for another composition, being close to the native TLS framework structure. A promising approach is to relax the key exchange functionality in the form that it outputs the session keys and two random values in the space of the finished

Protocol π

1. Upon activation with query (“establish-key”, $\texttt{SID}$, ID_I), where $ID_I{=}(\bot, I)$, the initiator sends the init message (r_I) where $r_I \overset{r}{\leftarrow} \{0,1\}^{p_1(k)}$ is a nonce. Upon activation with query (“establish-key”, $\texttt{SID}$, R), the responder waits for the receipt of the init message. It responds with own nonce $r_R \overset{r}{\leftarrow} \{0,1\}^{p_2(k)}$ and initializes a copy of $\mathcal{F}_{\text{KE}}^{(1,2)}$ with session identifier $\texttt{SID}_{\text{KE}}{=}(r_I|r_R)$ by sending query (“establish-key”, $\texttt{SID}_{\text{KE}}$, R) to $\mathcal{F}_{\text{KE}}^{(1,2)}$.

2. Upon receiving the response message, the initiator calls $\mathcal{F}_{\text{KE}}^{(1,2)}$ with session identifier $\texttt{SID}_{\text{KE}}{=}(r_I|r_R)$ on query (“establish-key”, $\texttt{SID}_{\text{KE}}$, ID_I) and waits for the delivery of output (“Key”, $\texttt{SID}_{\text{KE}}$, R, μ). It then computes the session keys $(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \texttt{PRF}_\mu(l_2)$ and the finished value $F_I \leftarrow \texttt{PRF}_\mu(l_3)$. Additionally, the initiator sends the final initiator message $(\texttt{E}_{k_e^I}(F_I|\texttt{HMAC}_{k_a^I}()))$.

3. When the responder receives the final initiator message (α), it first waits for the delivery of (“Key”, $\texttt{SID}_{\text{KE}}$, ID_I, μ) from $\mathcal{F}_{\text{KE}}^{(1,2)}$. Then, the responder computes in the same way the session keys $(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \texttt{PRF}_\mu(l_2)$. It decrypts the final initiator message $(F_I|t_I) \leftarrow \texttt{D}_{k_e^I}(\alpha)$ and verifies that $F_I \leftarrow \texttt{PRF}_\mu(l_3)$ and $t_I \leftarrow \texttt{HMAC}_{k_a^I}(F_I)$. If the verification fails, it aborts. Otherwise, it computes the finished value $F_R \leftarrow \texttt{PRF}_\mu(l_4)$ and sends the final responder message $(\texttt{E}_{k_e^R}(F_R|\texttt{HMAC}_{k_a^R}(F_R)))$. Finally, the responder terminates with local output (“Key”, $\texttt{SID}$, ID_I, κ) for $\kappa{=}(k_e^I, k_a^I, k_e^R, k_a^R)$.

4. Upon delivery of the final responder message (β), the initiator decrypts the message $(F_R|t_R) \leftarrow \texttt{D}_{k_e^R}(\beta)$. Then, it verifies that $F_R \leftarrow \texttt{PRF}_\mu(l_4)$ and $t_R \leftarrow \texttt{HMAC}_{k_a^R}(F_R)$. If the verification fails, it aborts. Otherwise, the initiator locally outputs (“Key”, $\texttt{SID}$, R, κ) for $\kappa{=}(k_e^I, k_a^I, k_e^R, k_a^R)$.

Figure 6: The TLS Handshake Protocol Structure, in the $\mathcal{F}_{\text{KE}}^{(1,2)}$-Hybrid model. **Note**, the finished values are the first messages, which are encrypted and authenticated with the session keys [18, Sect. 7.4.9.].

values, and next analyze whether a TLS-like handshake protocol without the commitment to the finished values securely emulates the relaxed functionality.

B Proofs for TLS Subroutines

B.1 Proof of Lemma 5

Subroutine DHE

1. Upon reception of an activation query (“establish-Key”, $\mathtt{SID}$, R), the responder chooses primes $p, q, q/p-1$ and g of order q in $\mathbb{Z}_p^*$. The DH exponent g^x is computed with $x \xleftarrow{r} \mathbb{Z}_q$. It calls $\mathcal{F}_{\mathrm{CERT}}$ with a message (“sign”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($\mathtt{SID}$, g, g^x)) where $\mathtt{SID}_{\mathrm{CERT0}}$=($R$, $\mathtt{SID} \circ \mathtt{0}$) contains the identity of the owner of the instance. The responder waits for the delivery of (“Signature”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($\mathtt{SID}$, g, g^x), σ). Finally, it computes the response message by placing the signature to the DH parameters (g, g^x, σ) and appends its identity R.
2. When receiving an activation query (“establish-Key”, $\mathtt{SID}$, $\perp$), the initiator waits for the delivery of the response message (g, g^x, σ, R). Then, it calls $\mathcal{F}_{\mathrm{CERT}}$ on query (“verify”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($g$, g^x), σ) and waits for the verification (“verified”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($g$, g^x), f). If σ is an invalid signature ($f=0$), it aborts. Otherwise, the initiator sends g^y with $y \xleftarrow{r} \mathbb{Z}_q$. Additionally, it computes the master secret $k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$ and locally outputs (“Key”, $\mathtt{SID}$, R, k_m).
3. Upon delivery of message (g^y), the responder derives the master secret $k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$ and locally outputs (“Key”, $\mathtt{SID}$, $\perp$, k_m).

Figure 7: Subroutine DHE, in the $\mathcal{F}_{\mathrm{CERT}}$-Hybrid model

Proof. We construct a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it interacts with subroutine DHE in front of adversary $\mathcal{A}$ and $\mathcal{F}_{\mathrm{KE}}^1$ in presence of $\mathcal{S}$. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and mimics an interaction with players executing DHE. Further, the simulator allows the adversary $\mathcal{A}$ to attack the simulated protocol execution in arbitrary way throughout the simulation. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$ and any output from $\mathcal{A}$ is copied to $\mathcal{S}$ in order to be forwarded to $\mathcal{Z}$. $\mathcal{S}$ tries to make the internal protocol simulation consistent with the real protocol execution under the condition that it has no information about the master key k_m. To this end, the simulator proceeds as follows:

1. **Simulating Activation of** R Upon receiving message (“establish-key”, $\mathtt{SID}$, R) from $\mathcal{F}_{\mathrm{KE}}^1$, $\mathcal{S}$ feeds $\mathcal{A}$ with the message (g, g^x, σ, R) from R. Here, x is chosen at random and σ is obtained by handing $\mathcal{A}$ the message (“sign”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($\mathtt{SID}$, g, g^x)) on behalf of $\mathcal{F}_{\mathrm{CERT}}$, and setting σ to the value returned by $\mathcal{A}$, where $\mathtt{SID}_{\mathrm{CERT0}}$=($R$, $\mathtt{SID} \circ \mathtt{0}$).
2. **Simulating Activation of** I Upon receiving message (“establish-key”, $\mathtt{SID}$, $\perp$) from $\mathcal{F}_{\mathrm{KE}}^1$, $\mathcal{S}$ waits for delivery of message (g, α, σ, P') from $\mathcal{A}$. It verifies that σ is a valid signature by querying $\mathcal{F}_{\mathrm{CERT}}$ on input (“verify”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($g$, α), σ). (The verification succeeds, if P'=R and σ is the response to a request by $\mathcal{S}$ of the form (“sign”, $\mathtt{SID}_{\mathrm{CERT0}}$, ($\mathtt{SID}$, g, α)) sent to $\mathcal{F}_{\mathrm{CERT}}$. It fails, if $P' \neq R$ is the identity of an uncorrupted party.) $\mathcal{S}$ mimics the response message by choosing y at random and sending g^y to R. In addition, it chooses the master key k_m to be a random value Δ_{k_m} in the same domain and sends (“Key”, $\mathtt{SID}$, $\perp$, k_m) to $\mathcal{F}_{\mathrm{KE}}^1$.

Functionality $\mathcal{F}_{\text{CERT}}$

$\mathcal{F}_{\text{CERT}}$ proceeds as follows, running on security parameter k, with parties P and an adversary $\mathcal{S}$. The `SID` is assumed to consist of a pair `SID`=(`PID`$_{\text{owner}}$, `SID`$'$), where `PID`$_{\text{owner}}$ is the owner of this instance.

Signature Generation: Upon receiving a value (“sign”, `SID`, m) from some the signer `PID`$_{\text{owner}}$, send (“sign”, `SID`, m) to the adversary. Upon receiving (“Signature”, `SID`, m, σ) from the adversary, verify that no entry (m, `SID`, 0) is recorded. If it is, then output an error message to $\mathcal{S}$ and halt. Else, output (“Signature”, `SID`, m, σ) to $\mathcal{S}$, and record the entry (m, σ, 1).

Signature Verification: Upon receiving a value (“verify”, `SID`, m, σ) from some party P, hand (“verify”, `SID`, m, σ) to the adversary. Upon receiving (“verified”, `SID`, m, φ) from the adversary, do:

1. If (m, σ, 1) is recorded then set f=1.
2. Else, if the signer is not corrupted, and no entry (m, σ', 1) for any σ' is recorded, then set f=0 and record the entry (m, σ, 0).
3. Else, if there is an entry (m, σ, f') recorded, then set f=f'.
4. Else, set f=φ, and record the entry (m, σ, φ).

Output (“verified”, `SID`, m, f) to P.

Figure 8: Certification Functionality

3. **Simulating Reception of the Response Message by** R When $\mathcal{A}$ delivers the message β, then two cases exists.
   - If $\beta \neq g^y$, then the adversary impersonated the client. $\mathcal{S}$ sets the master secret $k_m \leftarrow$ `PRF`$_{\beta^x}(l_1)$ and sends message (“impersonate”, `SID`, k_m) followed by a (“Key”, `SID`, R, k_m) message to $\mathcal{F}^1_{\text{KE}}$.
   - If $\beta = g^y$, then no impersonation occurred. $\mathcal{S}$ fixes the same master key $k_m = \Delta_{k_m}$ and answers $\mathcal{F}^1_{\text{KE}}$ with message (“Key”, `SID`, R, k_m).
4. **Simulating Static Corruption** If one of the parties is corrupted, then $\mathcal{S}$ proceeds by emulating a DHE protocol session, just as a honest party would play it. In particular, it fixes the master key $\tilde{\mu}$ as the adversary $\mathcal{A}$ says to do.

We now prove that the simulator $\mathcal{S}$ is such that no environment $\mathcal{Z}$ can distinguish between the ideal execution of $\mathcal{F}^1_{\text{KE}}$ and $\mathcal{S}$, and the real execution of DHE and $\mathcal{A}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model. This is done in the following way. We define an intermediate hybrid distributions $\mathcal{H}_{\mathbf{1}}$ and show that the environment notifies the change by contradicting the decisional Diffie-Hellman assumption. The hybrid distribution $\mathcal{H}_{\mathbf{1}}$ is defined as follows:

- $\mathcal{H}_{\mathbf{1}}$ takes the distribution of the output of $\mathcal{Z}$ which is identical to a real execution of $\mathcal{A}$ running DHE in the $\mathcal{F}_{\text{CERT}}$-hybrid model with the following exception. When protocol DHE instructs the initiator (respectively the responder) to evaluate the pseudo-random function `PRF`() with the premaster secret α^y (resp. β^x) in order to compute the master key k_m, we replace the output with an independently chosen random key from the same domain.

Claim 15 *Assume the Decisional Diffie-Hellman assumption holds, then* $\texttt{UC}-\texttt{EXEC}^{\mathcal{F}_{\mathrm{CERT}}}_{\mathrm{DHE},\mathcal{A},\mathcal{Z}} \approx \mathcal{H}_{\mathbf{1}}$.

Assume there is an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$ such that $\mathcal{Z}$ distinguishes with non-negligible probability between $\texttt{UC}-\texttt{EXEC}^{\mathcal{F}_{\mathrm{CERT}}}_{\mathrm{DHE},\mathcal{A},\mathcal{Z}}$ and $\mathcal{H}_{\mathbf{1}}$. We construct an adversary $\mathcal{D}$ that contradicts the Diffie-Hellman assumption. That is, given the tuple g^a, g^b, g^z where $a, b \overset{r}{\leftarrow} \mathbb{Z}_q$, $\mathcal{D}$ distinguishes between the case where $z{=}ab$ and $z \overset{r}{\leftarrow} \mathbb{Z}_q$.

Parameterized with the tuple g^a, g^b, g^z, $\mathcal{D}$ runs a copy of $\mathcal{Z}$ on a simulated interaction with $\mathcal{A}$ and parties running DHE in the $\mathcal{F}_{\mathrm{CERT}}$-hybrid model. $\mathcal{D}$ plays for $\mathcal{Z}$ the roles of $\mathcal{A}$ and the parties with the following exceptions. When the responder fixes g^x, it replaces the value with g^a. Next, if the initiator fixes g^y, it replaces the value with g^b. Further, whenever the players evaluate the pseudo-random function $\texttt{PRF}()$ on seed α^y (resp. β^x) in order to compute the master key, $\mathcal{D}$ replaces the evaluation with $\texttt{PRF}_{g^z}()$. Finally, whenever $\mathcal{Z}$ instructs $\mathcal{A}$ to corrupt a party, $\mathcal{D}$ outputs whatever the simulated $\mathcal{Z}$ outputs.

Consider the case where $z \overset{r}{\leftarrow} \mathbb{Z}_q$. We claim that in this case the view of the simulated $\mathcal{Z}$ is identically distributed to the view of $\mathcal{Z}$ in $\mathcal{H}_{\mathbf{1}}$. This is so because g^x, g^y and the seed for the evaluation of the pseudo-random function $\texttt{PRF}_{g^z}()$ are independently and randomly chosen, assuming that $\mathcal{D}$ did not abort. Now, consider the case where $z{=}ab$. We claim that in this case the view of the simulated $\mathcal{Z}$ is identically distributed to the view of $\mathcal{Z}$ in $\texttt{UC}-\texttt{EXEC}^{\mathcal{F}_{\mathrm{CERT}}}_{\mathrm{DHE},\mathcal{A},\mathcal{Z}}$. The only potential mismatch is that the initiator accepts a value α which is different from the value g^x the responder has chosen. Such a mismatch cannot occur due to the security properties of $\mathcal{F}_{\mathrm{CERT}}$, i.e. the initiator accepts α only when σ is a valid signature under the condition that $\mathcal{Z}$ did not instruct $\mathcal{A}$ to corrupt the responder. And the only way for this to occur is that the responder has registered as owner of the instance $\texttt{SID}_{\mathrm{CERT0}}$, and queried $\mathcal{F}_{\mathrm{CERT}}$ on message (“sign”, $\texttt{SID}_{\mathrm{CERT0}}$, ($\texttt{SID}$, g, g^x)) in order to retrieve a signature.

Claim 16 *Assume* $\texttt{PRF}()$ *is a secure pseudo-random function, then* $\mathcal{H}_{\mathbf{1}} \approx \texttt{UC}-\texttt{EXEC}_{\mathcal{F}^1_{\mathrm{KE}},\mathcal{S},\mathcal{Z}}$.

Assume that such environment $\mathcal{Z}$ that distinguishes with non-negligible simulation slice between the interaction $\mathcal{H}_{\mathbf{1}}$and $\texttt{UC}-\texttt{EXEC}_{\mathcal{F}^1_{\mathrm{KE}},\mathcal{S},\mathcal{Z}}$ exists. We construct an adversary $\mathcal{D}$ who has access to the black-box oracle $\mathcal{O}_{\mathcal{P}}$ that contains throughout the whole simulation either $\texttt{PRF}_{k_p}()$ for some randomly chosen secret k_p or a truly random function with the same range. $\mathcal{D}$ runs a copy of $\mathcal{Z}$ and emulates the interaction with parties running DHE in the $\mathcal{F}_{\mathrm{CERT}}$-hybrid model and $\mathcal{A}$ unless it is required to compute the master secret k_m for which it calls $\mathcal{O}_{\mathcal{P}}$. It is easy to see that $\mathcal{D}$ interpolates between $\mathcal{H}_{\mathbf{1}}$ (this is the case where $\mathcal{O}_{\mathcal{P}}$ contains $\texttt{PRF}_{k_p}()$) and $\texttt{UC}-\texttt{EXEC}_{\mathcal{F}^1_{\mathrm{KE}},\mathcal{S},\mathcal{Z}}$ (this is the case where $\mathcal{O}_{\mathcal{P}}$ contains the truly random function). Hence, the output of $\mathcal{Z}$ can be directly used by $\mathcal{D}$ to decide on the content of $\mathcal{O}_{\mathcal{P}}$.

■

B.2 Proof of Lemma 6

Proof. Again, we construct a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ distinguishes between the case that $\mathcal{Z}$ interacts with DHS in the $\mathcal{F}_{\mathrm{CA}}$-hybrid model and $\mathcal{A}$, and the case that $\mathcal{Z}$ interacts with $\mathcal{F}^1_{\mathrm{KE}}$ and $\mathcal{S}$. $\mathcal{S}$ runs a simulated copy of the adversary $\mathcal{A}$ and mimics for $\mathcal{A}$ an interaction with players executing DHS under the limitation that it has no clue about the master key k_m. It

Subroutine DHS

1. Upon reception of an activation query ("establish-Key", $\texttt{SID}$, R), the responder sends a response messages consisting of its identity R, signaling that it has registered static DH parameters at $\mathcal{F}_{\text{CA}}$ by sending a query ("register", $\texttt{SID}_{\text{CA0}}$, (g, g^x)) where $\texttt{SID}_{\text{CA0}}$=$(R, \texttt{SID} \circ \texttt{0})$ includes its identity.
2. When the initiator receives an activation query ("establish-Key", $\texttt{SID}$, $\perp$), it waits for the delivery of the response message (R) and invokes $\mathcal{F}_{\text{CA}}$ with query ("retrieve", $\texttt{SID}_{\text{CA0}}$). Then, the initiator waits for the output ("retrieve", $\texttt{SID}_{\text{CA0}}$, (g, g^x)). If no value is retrievable, it aborts. Otherwise, the initiator sends g^y with $y \overset{r}{\leftarrow} \mathbb{Z}_q$ and computes the master secret $k_m \leftarrow \texttt{PRF}_{g^{xy}}(l_1)$. It terminates the protocol with local output ("Key", $\texttt{SID}$, R, k_m).
3. Upon delivery of message (g^y), the responder computes the master secret $k_m \leftarrow \texttt{PRF}_{g^{xy}}(l_1)$ and locally outputs ("Key", $\texttt{SID}$, $\perp$, k_m).

Figure 9: Subroutine DHS, in the $\mathcal{F}_{\text{CA}}$-Hybrid model

allows $\mathcal{A}$ to attack the simulated protocol execution in arbitrary way throughout the simulation. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$ and any output from $\mathcal{A}$ is copied to $\mathcal{S}$ in order to be forwarded to $\mathcal{Z}$. In order to make the environment's view consistent with its view of the real-world protocol execution, $\mathcal{S}$ proceeds as follows:

1. **Simulating Activation of** R Upon receiving message ("establish-key", $\texttt{SID}$, R) from $\mathcal{F}^1_{\text{KE}}$, $\mathcal{S}$ feeds $\mathcal{A}$ with the message ("register", $\texttt{SID}_{\text{CA0}}$, (g, g^x)), where $\texttt{SID}_{\text{CA0}}$=$(R, \texttt{SID} \circ \texttt{0})$ includes the responder's identity. It simulates that the responder has registered at $\mathcal{F}_{\text{CA}}$. In addition, it feeds $\mathcal{A}$ with message (R) from R.

2. **Simulating Activation of** I Upon receiving message ("establish-key", $\texttt{SID}$, $\perp$) from $\mathcal{F}^1_{\text{KE}}$, $\mathcal{S}$ waits for delivery of message (P') from $\mathcal{A}$. It retrieves the registered parameters by emulating a request ("retrieve", $\texttt{SID}_{\text{CA0}}$), where $\texttt{SID}_{\text{CA0}}$=$(P', \texttt{SID} \circ \texttt{0})$, to $\mathcal{A}$ on behalf of I. It waits for the answer ("retrieve", $\texttt{SID}_{\text{CA0}}$, (g, g^x)) from $\mathcal{A}$. (The retrieval succeeds, if P'=R and (g, g^x) is the response to a the previous registration request by $\mathcal{S}$ of the form ("register", $\texttt{SID}_{\text{CA0}}$, (g, g^x)) sent to $\mathcal{F}_{\text{CERT}}$. It fails, if $P' \neq R$ is the identity of an uncorrupted party.) In addition, $\mathcal{S}$ simulates the response message by choosing y at random and sending g^y to R. Finally, it chooses the master key k_m to be a random Δ_{k_m} in the same range and sends ("Key", $\texttt{SID}$, $\perp$, k_m) to $\mathcal{F}^1_{\text{KE}}$.

3. **Simulating Reception of the Response Message by** R When $\mathcal{A}$ delivers the message β, then two cases exists as in the previous simulation.

   - If $\beta \neq g^y$, then the adversary acts as the initiator. $\mathcal{S}$ reacts in the following way to make the simulation consistent with the real-world. It sets the master secret $k_m \leftarrow \texttt{PRF}_{\beta^x}(l_1)$ and sends message ("impersonate", $\texttt{SID}$, k_m) followed by a ("Key", $\texttt{SID}$, R, k_m) message to $\mathcal{F}^1_{\text{KE}}$.
   - If β=g^y, then the adversary did not intercept the communication between the initiator and responder. $\mathcal{S}$ fixes the same master key k_m=Δ_{k_m} as in the previous simulation step and queries $\mathcal{F}^1_{\text{KE}}$ with message ("Key", $\texttt{SID}$, R, k_m).

Functionality $\mathcal{F}_{\text{CA}}$

$\mathcal{F}_{\text{CERT}}$ proceeds as follows, running on security parameter k, with parties P and an adversary $\mathcal{S}$. The `SID` is assumed to consist of a pair `SID`=(`PID`$_{\text{owner}}$, `SID`$'$), where `PID`$_{\text{owner}}$ is the owner of this instance.

Registration: Upon receiving the first message (“register”, `SID`, m) from party P, send (“register”, `SID`, m) to the adversary; upon receiving `ok` from the adversary, and if this is the first request from P, then record the pair (P, m).

Retrieval: Upon receiving a message (“retrieve”, `SID`) from other party P', send (“retrieve”, `SID`, P') to the adversary, and wait for an `ok` from the adversary. Then, if there is a recorded pair (`SID`, m) output (“retrieve”, `SID`, m) to P'. Else output (“retrieve”, `SID`, $\perp$) to P'.

Figure 10: Certificate Authority Functionality

4. **Simulating Static Corruption** If one of the parties gets corrupted, then $\mathcal{S}$ proceeds by emulating a DHS protocol session, just as a honest party would play it. In particular, it fixes the master key $\tilde{\mu}$ as the adversary $\mathcal{A}$ says to do.

Analyzing $\mathcal{S}$, it can be seen that the environment’s view when interacting with DHS is computational indistinguishable from its view when interacting with DHE. The only difference is that DHS calls $\mathcal{F}_{\text{CA}}$ to fix g^x (instead of $\mathcal{F}_{\text{CERT}}$). Thus, the potential mismatch is that the initiator accepts a value α which is different from the value g^x the responder has fixed. However, such a mismatch cannot occur due to the security properties of $\mathcal{F}_{\text{CA}}$. Hence, by the reduction to the Diffie-Hellman assumption and the security of the pseudo-random function, it can be seen that both views of $\mathcal{Z}$ are computational indistinguishable.

Claim 17 *Assume the Diffie-Hellman assumption holds and* `PRF()` *is a secure pseudo-random function, then* $\texttt{UC-EXEC}^{\mathcal{F}_{\text{CA}}}_{\text{DHS},\mathcal{A},\mathcal{Z}} \approx \texttt{UC-EXEC}_{\mathcal{F}^1_{\text{KE}},\mathcal{S},\mathcal{Z}}$.

See the proof of hybrid $\mathcal{H}_1$ for subroutine DHE. ■

B.3 Proof of Lemma 7

Proof. Let $\mathcal{A}$ be a real-world adversary that operates against EKT. As in the prior simulations, we construct an ideal-world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between the case that it interacts with $\mathcal{A}$ and parties running EKT or with $\mathcal{S}$ in the ideal world for $\mathcal{F}^1_{\text{KE}}$. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and mimics an interaction with players executing EKT. It tries to make the internal protocol simulation consistent with the real protocol execution. The simulator allows the adversary $\mathcal{A}$ to attack the simulated protocol execution in arbitrary way throughout the simulation. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$ and any output from $\mathcal{A}$ is copied to $\mathcal{S}$ in order to be forwarded to $\mathcal{Z}$. In detail, $\mathcal{S}$ proceeds in the following way:

1. **Simulating Activation of** R Upon receiving message (“establish-key”, `SID`, R) from $\mathcal{F}^1_{\text{KE}}$, $\mathcal{S}$ feeds $\mathcal{A}$ with the message (R), indicating that the responder has registered at $\mathcal{F}_{\text{CPKE}}$.

Subroutine EKT

1. Upon reception of an activation query (“establish-Key”, $\texttt{SID}$, R), the responder sends the response message (R), indicating that it owns an instance of $\mathcal{F}_{\text{CPKE}}$.
2. Upon reception of an activation query (“establish-Key”, $\texttt{SID}$, $\perp$), the initiator waits for the delivery of the response message (R). It fixes a premaster secret $k_p \overset{r}{\leftarrow} \{0,1\}^{p_3(k)}$. Next, the initiator sends the value (“encrypt”, $\texttt{SID}_{\text{CPKE0}}$, k_p) to $\mathcal{F}_{\text{CPKE}}$ where $\texttt{SID}_{\text{CPKE0}}$=($R$, $\texttt{SID} \circ \texttt{0}$) contains the owner identity, and waits for the delivery of (“Ciphertext”, $\texttt{SID}_{\text{CPKE0}}$, c). Then, the initiator derives master secret $k_m \leftarrow \texttt{PRF}_{k_p}(l_1)$, forwards ciphertext value (c) to the responder, and generates local output (“Key”, $\texttt{SID}$, R, k_m).
3. Upon delivery, the responder first decrypts c by sending (“decrypt”, $\texttt{SID}_{\text{CPKE0}}$, c) to $\mathcal{F}_{\text{CPKE}}$ and waits for the output (“Plaintext”, $\texttt{SID}_{\text{CPKE0}}$, k_p). If the decryption succeeds, the responder computes the master secret $k_m \leftarrow \texttt{PRF}_{k_p}(l_1)$ and terminates with local output (“Key”, $\texttt{SID}$, $\perp$, k_m). Otherwise, the responder aborts.

Figure 11: Subroutine EKT, in the $\mathcal{F}_{\text{CPKE}}$-Hybrid model

2. **Simulating Activation of** I Upon receiving message (“establish-key”, $\texttt{SID}$, $\perp$) from $\mathcal{F}^1_{\text{KE}}$, $\mathcal{S}$ waits for delivery of message (P') from $\mathcal{A}$. It chooses a premaster secret k_p at random and queries $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{CPKE}}$ with message (“encrypt”, $\texttt{SID}_{\text{CPKE0}}$, k_p), where $\texttt{SID}_{\text{CPKE0}}$=($P'$, $\texttt{SID} \circ \texttt{0}$). (The request succeeds unless P' is an uncorrupted party and P'=R. Otherwise, the simulator aborts.) Upon reception of message (“Ciphertext”, $\texttt{SID}_{\text{CPKE0}}$, c), $\mathcal{S}$ sets the master key k_m to be a value Δ_{k_m} in the same range, forwards the ciphertext c to the responder and outputs (“Key”, $\texttt{SID}$, $\perp$, k_m) to $\mathcal{F}^1_{\text{KE}}$.
3. **Simulating Reception of the Response Message by** R When $\mathcal{A}$ delivers the message α, $\mathcal{S}$ mimics the decryption of α by sending (“decrypt”, $\texttt{SID}_{\text{CPKE0}}$, α) to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{CPKE}}$ and waiting for the response (“Plaintext”, $\texttt{SID}_{\text{CPKE0}}$, k'_p). Now, two cases exists agai.
   - If k'_p=k_p, then $\mathcal{S}$ fixes the same master key k_m=Δ_{k_m} as in the previous simulation step and outputs (“Key”, $\texttt{SID}$, $\perp$, k_m) to $\mathcal{F}^1_{\text{KE}}$.
   - Otherwise, $\mathcal{S}$ sets the master secret $k_m \leftarrow \texttt{PRF}_{k'_p}(l_1)$ and outputs message (“impersonate”, $\texttt{SID}$, k_m) followed by a (“Key”, $\texttt{SID}$, R, k_m) message to $\mathcal{F}^1_{\text{KE}}$.
4. **Simulating Static Corruption** If one of the parties gets corrupted, then $\mathcal{S}$ proceeds by emulating a EKT protocol session, just as a honest party would play it. In particular, it fixes the master key $\tilde{\mu}$ as the adversary $\mathcal{A}$ says to do.

We now prove that no environment $\mathcal{Z}$ exists that distinguishes with non-negligible simulation slice between the case that $\mathcal{Z}$ interacts with EKT and $\mathcal{A}$, and the case that $\mathcal{Z}$ interacts with $\mathcal{F}^1_{\text{KE}}$ and $\mathcal{S}$. This is done by reduction to the security of the pseudo-random function.

Claim 18 *Assume* $\texttt{PRF}()$ *is a secure pseudo-random function, then* $\texttt{UC}-\texttt{EXEC}^{\mathcal{F}_{\text{CPKE}}}_{\text{EKT},\mathcal{A},\mathcal{Z}} \approx \texttt{UC}-\texttt{EXEC}_{\mathcal{F}^1_{\text{KE}},\mathcal{S},\mathcal{Z}}$.

Assume that such environment $\mathcal{Z}$ that distinguishes with non-negligible simulation slice between the interaction $\texttt{UC}-\texttt{EXEC}^{\mathcal{F}_{\text{CPKE}}}_{\text{EKT},\mathcal{A},\mathcal{Z}}$ and $\texttt{UC}-\texttt{EXEC}_{\mathcal{F}^1_{\text{KE}},\mathcal{S},\mathcal{Z}}$ exists. We construct an adversary $\mathcal{D}$ who has

Functionality $\mathcal{F}_{\text{CPKE}}$

$\mathcal{F}_{\text{CPKE}}$ proceeds as follows, when parameterized by message domain M, a function E with domain M and range $\{0,1\}^*$, and function D of domain $\{0,1\}^*$ and range $M \cup \mathtt{error}$. The $\mathtt{SID}$ is assumed to consist of a pair $\mathtt{SID}{=}(\mathtt{PID}_{\text{owner}}, \mathtt{SID}')$, where $\mathtt{PID}_{\text{owner}}$ is the owner of this instance.

Encryption: Upon receiving a value ("encrypt", $\mathtt{SID}$, m) from a party P, proceed as follows:

1. If $m \notin M$ then return an error message to P.
2. If $m \in M$ then:
   - If the owner of this instance of $\mathcal{F}_{\text{CPKE}}$ is corrupted, then hand also the entire value m to the adversary and receive tag c.
   - Else, calculate a value c by choosing a random r of the same length as m, and selecting $c \leftarrow E(r)$.

   Record the pair (c, m), and output ("Ciphertext", $\mathtt{SID}$, c). (If c already appears in a previously recorded pair then return an error message to P.)

Decryption: Upon receiving a value ("decrypt", $\mathtt{SID}$, c) from the owner of this instance, proceed as follows. (If the input is received from another party then ignore.)

1. If there is a recorded pair (c, m), then hand ("Plaintext", $\mathtt{SID}$, m) to P.
2. Otherwise, compute $m \leftarrow D(c)$, and hand ("Plaintext", $\mathtt{SID}$, m) to P.

Figure 12: Certified Public Key Encryption Functionality

access to the black-box oracle $\mathcal{O}_{\mathcal{P}}$. $\mathcal{D}$ plays for $\mathcal{Z}$ the roles of $\mathcal{A}$ and the parties running EKT. The black-box oracle $\mathcal{O}_{\mathcal{P}}$ contains throughout the whole simulation either $\mathtt{PRF}_{k_p}()$ for some randomly chosen secret k_p or a truly random function with the same range. $\mathcal{D}$ runs a copy of $\mathcal{Z}$ and emulates the interaction with parties running EKT in the $\mathcal{F}_{\text{CPKE}}$-hybrid model and $\mathcal{A}$ unless it is required to compute the master secret k_m for which it calls $\mathcal{O}_{\mathcal{P}}$.

Consider the case where $\mathcal{O}_{\mathcal{P}}$ calls the truly random function. We claim that in this case the environment's view is identically distributed to $\mathtt{UC{-}EXEC}_{\mathcal{F}^1_{\text{KE}}, \mathcal{S}, \mathcal{Z}}$. This is so because the master key is chosen randomly and independently from the initiator's state under the condition that $\mathcal{S}$ did not abort. Now, consider the case where $\mathcal{O}_{\mathcal{P}}$ calls $\mathtt{PRF}_{k_p}()$. We claim that in this case the environment's view is identically distributed to $\mathtt{UC{-}EXEC}^{\mathcal{F}_{\text{CPKE}}}_{\text{EKT}, \mathcal{A}, \mathcal{Z}}$. Indeed, the only potential mismatch is that the responder accepts a premaster key k_p that has not been fixed by the initiator, assuming $\mathcal{Z}$ neither instructed $\mathcal{A}$ to corrupt nor impersonate the initiator. Such a mismatch cannot occur due to the security of $\mathcal{F}_{\text{CPKE}}$. The responder accepts the premaster key k_p, when it receives output ("Plaintext", $\mathtt{SID}_{\text{CPKE0}}$, k_p) from $\mathcal{F}_{\text{CPKE}}$. And the only way for this to occur is that the initiator has uploaded the premaster key by sending ("encrypt", $\mathtt{SID}_{\text{CPKE0}}$, k_p) to $\mathcal{F}_{\text{CPKE}}$ before. ■