(De-)Constructing TLS

Markulf Kohlweiss[1], Ueli Maurer[2], Cristina Onete[3], Björn Tackmann[2], and Daniele Venturi[*4]

[1]*Microsoft Cambridge*
[2]*ETH Zürich*
[3]*IRISA/INRIA, Univ. Rennes 1*
[4]*Sapienza University of Rome*

January 7, 2014

Abstract

One of the most important applications of cryptography is the establishment of secure communication channels between two entities (e.g. a client and a server), and the protocol most widely used for this purpose is TLS.

A key goal of research in cryptography is to provide security proofs for cryptographic protocols. This task is particularly difficult if the considered protocol has not been designed with provable security in mind, as is the case for TLS.

Results on provable security differ with respect to (1) the assumptions made and (2) the statement that is proved to follow from the assumptions. It is important that the proved statement is of a form that allows for both comparisons of protocol performance, and for direct use in the proof of a higher-level protocol. Security statements should thus be exact (as opposed to asymptotic), giving precise upper bounds for the security level guaranteed by a protocol. Furthermore, a key to analyzing and designing cryptographic protocols is a modularization in which the role of each cryptographic primitive (e.g. encryption) or mechanism (e.g. nonce exchange) is made explicit, and the security of its application is proved in isolation, once and for all. The constructive cryptography framework provides a sound instantiation of this approach. A modular step constructs a specific resource from certain (assumed) resources, and the overall protocol is the composition of several such construction steps. The security proof for the overall protocol follows directly from the composition theorem as well as the individual (reasonably simple) security proofs for the modules. Moreover, the actual security statement for the overall protocol is of a standardized form, in terms of a resource, which makes it straight-forward to use the protocol in a higher-level context, with the overall security proof again following from the composition theorem.

In this paper, we provide such a constructive treatment of TLS. We provide a deconstruction of TLS into modular steps and a security proof for each step which, compared to previous work, results in the above mentioned advantages. For the key-exchange step in particular, we analyze the RSA-based and both Diffie-Hellman-based variants (with static and ephemeral server key) under a non-randomizability assumption for RSA-PKCS and the Gap Diffie-Hellman assumption, respectively; in all cases we make use of random oracles. In general, since the design of TLS is not modular, the constructive decomposition is less fine-grained than one might wish to have and than it is for a modular design. This paper therefore also suggests new insights into the intrinsic problems incurred by a non-modular protocol design such as that of TLS.

[*]Part of the work done while at Aarhus University supported by the Danish Council for Independent Research via DFF Starting Grant 10-081612.

Contents

1 Introduction

Initially developed by Netscape as the Secure Socket Layer (SSL) protocol [Hic95], the SSL/TLS protocol family aims to provide end-to-end security for bilateral communication over the Internet. The original protocol suffered from several vulnerabilities; this led to the development of a number of subsequent protocol versions, each one fixing flaws discovered in the previous version. The most recent protocol version is known as Transport Layer Security (TLS) version 1.2 [DR08]. While the initial protocol was developed for protecting HTTP connections between a browser and a web server, many current Internet protocols including, e.g., SMTP or IMAP for transmitting e-mails and LDAP for accessing directories, have been extended to allow for securing the transmission with TLS. Because of its practical importance, the security of SSL/TLS has attracted a lot of attention in the literature, see e.g. [WS96, Pau99, Kra01, CK02, JK02, Bar04, HSD+05, MSW08, MT10, PRS11, AP12, BFCZ12, FHM+12, GIJ+12, JKSS12, ABP+13, AP13, BFK+13a, BFS+13, GKS13, KSS13, KPW13], in chronological order.

1.1 Overview and Previous Work

The TLS protocol consists of two parts, the *handshake*—essentially a key-exchange protocol that can be used with either unilateral or mutual authentication—and the *record protocol*—which protects the transmission of application data using the key obtained during the handshake. The TLS handshake offers several alternative key-exchange methods based on different cryptographic primitives. In each session, the actual method is chosen depending on the implementation, the available public keys, and the configuration. There are three standard methods: based on RSA encryption, on a static certified Diffie-Hellman key, or on signatures and an (ephemeral) Diffie-Hellman exchange, respectively. The typical setting is to only authenticate the server to the client, but client authentication is also possible if the client has a certified public key.

While a sequence of results about the record protocol [Kra01, MT10, PRS11] provides a comprehensive treatment thereof, the handshake protocol is not as well-understood. The reason is that the protocol was not designed with provable security in mind (see details in Section 1.2); it is inherently non-modular and uses cryptographic primitives in non-standard ways, which severely complicates security proofs. One particular observation with respect to non-modularity appearing in many analyses is that the final messages of the handshake already *use* the keys that are agreed upon; a fact that prohibits an analysis of the full handshake in common security models (e.g., [BR93, CK01, CK02]). Beyond that, because of the non-standard use of schemes and primitives, papers that analyze (parts of) TLS must generally choose between analyzing a modified version of the protocol, analyzing the original protocol in idealized models (such as the random oracle model), or using tailor-made computational assumptions.

Proving modified protocols was an early approach towards obtaining an intuition of the security of the TLS handshake. For example, [MSW08, JK02] drop the final message of the handshake. A more extreme approach is taken by Gajek et al. [GMP+08], who modify several details to achieve security in the UC framework [Can01]. (As pointed out by Küsters et al. [KT11] this additionally requires the insertion of session identifiers into the protocol. Küsters et al. also give a case study and describe how one could model TLS more faithfully in the IITM [KT11] model.) However, extending the security proof of a *modified* protocol to that of the *real* protocol is usually hard; for SSL/TLS, it is nearly impossible.

Recent game-based analyses of TLS made important steps towards assessing the security of the true protocol. Jager et al. [JKSS12] treated the security of a specific configuration for the key-exchange step, i.e. TLS-DHE. In their (game-based) analysis, Jager et al. consider both the TLS handshake and the record protocol to be executed together. By treating the last message of the handshake as the first message of the record protocol, they circumvent the problems related to using one of the extracted keys for encryption during the key-exchange protocol. They showed that the TLS handshake and the record protocol *together* achieve the tailor-made (game-based) security notion "authenticated and confidential channel establishment", under a specific, new security assumption. A (slightly) more modular analysis is given by Brzuska et al. [BFS$^+$13], who view the TLS handshake and record protocol as a composition of a key agreement protocol and a scheme for achieving secure communication. They weaken the usual key-exchange security definitions and show explicitly that, under a specific notion of composition, the two parts can be proved secure together, in a meaningful way. Thus, this approach enables a security proof with some degree of modularity.

The recent work of Krawczyk, Paterson, and Wee [KPW13] extends the analysis of TLS-DHE [JKSS12] to other handshake configurations, like TLS-DH and TLS-RSA, and it is to our knowledge the most comprehensive analysis of TLS to date. Their approach is based on modeling the TLS handshake by means of a Key Encapsulation Mechanism (KEM). They also describe how to instantiate the KEM to capture each configuration. Subsequently, they show that, if the KEM attains a security notion known as Constrained CCA security (CCCA), the combined protocol consisting of KEM and record layer attains the ACCE security notion introduced by Jager et al. [JKSS12].

Other approaches have been taken to analyzing the security of TLS. For example, the symbolic model approach of [Pau99, HSD$^+$05, BFCZ12]. Recently, [BFK$^+$13a] moved away from symbolic models and gave a meaningful formal statement about a standard compliant implementation of TLS called miTLS. They describe the cryptographic idealizations of the cryptographic modules of TLS which are indistinguishable by typed adversaries. This allows for a type-based and cryptographically faithful verification of miTLS.

Previous results in constructive cryptography. Some related schemes have already been analyzed following the constructive cryptography paradigm [MR11, Mau11]. Maurer and Tackmann [MT10] used this approach to analyze the TLS record protocol as a construction of a secure channel from a shared key and insecure communication channels. Recently, Maurer, Tackmann, and Coretti [MTC13] described a unilateral key-exchange protocol in this model; the goal of their protocol is similar to the goal of certain configurations of TLS (without client authentication); their protocol is considerably simpler and secure under weaker assumptions, but does not allow to adaptively choose cipher suites.

1.2 Contributions of this Paper

In this paper, we define and prove the security of TLS following the constructive cryptography paradigm of Maurer and Renner [MR11]. We show that TLS/SSL with unilateral authentication *constructs* a *unilaterally secure communication channel* from an insecure communication network such as the Internet, a public-key infrastructure, and a public random oracle (an idealization of a hash function).

We de-compose the TLS protocol into a sequence of sub-protocols that each achieve a smaller construction step. Each step is then achieved by one or more cryptographic schemes (like key-exchange mechanisms, key confirmation, symmetric encryption, and MAC), or by security mechanisms (like nonces). The goal of each scheme or mechanism is to *construct* an ideal resource, which then serves as an assumed resource in the next step. The composition of all steps yields the complete TLS protocol, and the composition theorem guarantees the soundness of this approach.

This method of analyzing the TLS protocol has two main advantages:

1. The composition theorem (cf. [MR11, MT10, KMO$^+$13]) guarantees that the TLS protocol

can be used, given the assumed resources, whenever an application or higher-level protocol requires the type of secure channel described in our definitions.

2. As each step is proven in isolation and the steps are combined generically by the composition theorem,
   (a) the analysis of each individual step becomes simpler, because it can be performed independently of the other steps;
   (b) if an alternative security analysis of one step is given, then one can immediately use this proof without adapting the proofs of the remaining primitives. In particular, by adapting the techniques of [JKSS12, KPW13] to prove the key-exchange steps without a random oracle, we can immediately obtain a standard-model analysis of the complete protocol without re-proving the remaining steps;
   (c) further optional sub-protocols, like other symmetric ciphers or elliptic curve methods, are integrated into our analysis by proving simply that they construct a specific step. The security of the full protocol will then follow generically.

Unfortunately, TLS tries hard to resist this modularization, and it also uses many techniques heuristically: the cryptographic keys intended to protect the payload are also used for the confirmation (here called "finished") messages, the pseudo-random function is keyed with a non-uniform key, and various protocol messages are unnecessarily included in the confirmation messages, to name a few. These unfortunate design choices complicate our analysis: sometimes abstractions cannot be introduced at the intuitively appealing levels, often "auxiliary" data must be passed through multiple protocol layers, and as the potential interference of different protocol sessions is resolved only at a late stage, many protocol steps must be analyzed in a complicated "multi-session" scenario.

Furthermore, it appears impossible to perform a reasonably modular analysis for the RSA-PKCS variant of the handshake under the OW-PCA assumption [JK02], even in the random oracle model. This is evidence that the not very modular proof of [KPW13] for CCCA security of the TLS key extraction mechanism is hard to avoid unless one is willing to make additional assumptions. Building on a recent result by [BFK$^+$13b], we show that a reasonably weak non-randomizability assumption that they call NR-PCA is sufficient for a more modular constructive proof. We describe these problems in more detail in Section 3.3.3.

Although various of the above mentioned obstacles could have been bypassed by modifying the scheme, we kept our analysis close to the realistic TLS protocol. The result is a more complicated analysis; however, the artifices are immanent to the protocol, not to our technique.

In total, we analyze the three main methods for the establishment of the master secret in the handshake, namely TLS-DH, TLS-DHE, and TLS-RSA, as well the two main methods for protecting the payload messages in the record layer protocol, namely the Authenticate-then-Encrypt combinations using a stream cipher resp. a block cipher in CBC-mode, and a MAC. By the composition theorem, we obtain the security of all possible combinations. We use the GapDH assumption for the Diffie-Hellman based cipher suites, the NR-PCA assumption for RSA PKCS#7, pseudo-randomness for the stream and block ciphers, and strong unforgeability for the MAC. Moreover, we require the hash function used in the handshake to be collision-resistant, and our proofs of the handshake protocols are based on the random oracle assumption.

1.3 Outline

We now describe, at a high level, the cryptographic mechanisms used in TLS and the constructive steps they achieve. We start with a simplified description of the protocol in Figure 1, where the notation $\langle m \rangle_\kappa$ denotes the encryption of a message m, possibly together with other messages, under the key κ.

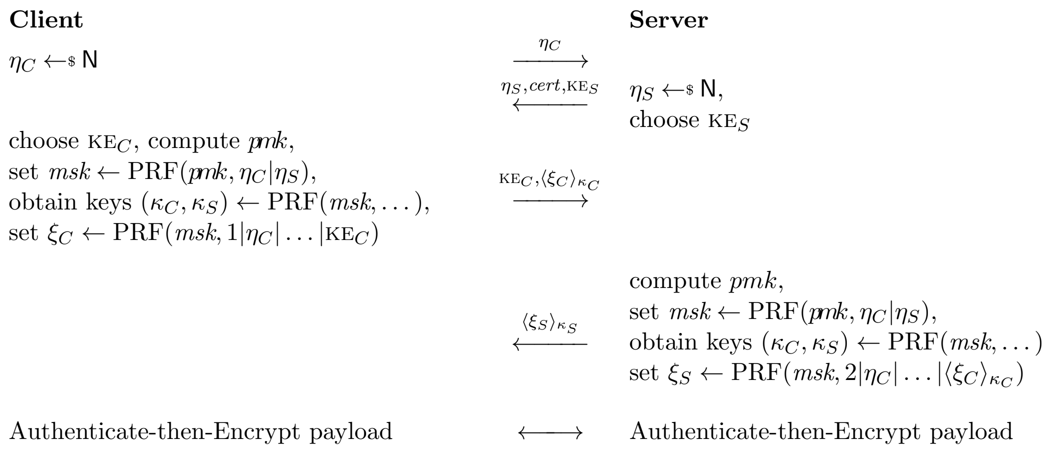
Client
Server
$\eta_C \leftarrow_\$ \mathsf{N}$
η_C
$\eta_S, cert, \textsc{ke}_S$
$\eta_S \leftarrow_\$ \mathsf{N}$,
choose $\textsc{ke}_S$
choose $\textsc{ke}_C$, compute pmk,
set $msk \leftarrow \mathrm{PRF}(pmk, \eta_C|\eta_S)$,
obtain keys $(\kappa_C, \kappa_S) \leftarrow \mathrm{PRF}(msk, \dots)$,
set $\xi_C \leftarrow \mathrm{PRF}(msk, 1|\eta_C|\dots|\textsc{ke}_C)$
$\textsc{ke}_C, \langle\xi_C\rangle_{\kappa_C}$
compute pmk,
set $msk \leftarrow \mathrm{PRF}(pmk, \eta_C|\eta_S)$,
obtain keys $(\kappa_C, \kappa_S) \leftarrow \mathrm{PRF}(msk, \dots)$,
set $\xi_S \leftarrow \mathrm{PRF}(msk, 2|\eta_C|\dots|\langle\xi_C\rangle_{\kappa_C})$
$\langle\xi_S\rangle_{\kappa_S}$
Authenticate-then-Encrypt payload
Authenticate-then-Encrypt payload

Figure 1: The TLS handshake as a message exchange

Decomposition into sub-protocols. At a high level, our analysis proceeds by repeatedly "scraping off" a part of the protocol, starting at the beginning; at each step, we consider the remainder of the protocol as the "payload" of the part we analyze. This permits us to then use the composition theorem to conclude the security of the entire protocol. We proceed in the following steps.

1. Nonce generation. Each client chooses a nonce $\eta_C \leftarrow_\$ \mathsf{N}$ for a distribtion N on the set of nonces.

Since TLS is often used in a unilateral mode where a client has no distinguished name, the client nonce can be seen as a "disposable name" that is (with high probability) unique to that client—the purpose of this nonce is to separate protocol sessions of different clients.

This guarantee is formalized as a resource NAME, which provides a unique name to each client.

2. Nonce exchange. The client transmits the nonce η_C (which it obtained from the resource NAME). The server chooses a nonce $\eta_S \leftarrow_\$ \mathsf{N}$ and sends it to the client.

The server uses the client nonce to identify the client—note that we focus on the unilateral case where the client is not authenticated. The server nonce is not crucial in our analysis, but to guarantee in the key derivation step (for which we make a random oracle assumption) that the data in each session is unique, we assume uniqueness of the server nonce, losing a collision term. However, if one analyzes the key derivation in more detail and not in the random oracle model, the entropy contained in the server nonce may be a valuable input for extraction.

This guarantee is formalized as a resource SNET that outputs nonces (potentially chosen adversarially) and associates them with the (fully) insecure communication channels giving no guarantee of consistency.

3. Unilateral key exchange. The server chooses a key-exchange message $\textsc{ke}_S$ according to the chosen cipher suite (the message may be empty or a Diffie-Hellman element), and sends its certificate *cert* and $\textsc{ke}_S$ to the client. The client also chooses a key-exchange message $\textsc{ke}_C$ accordingly and sends it to the server. Both compute the pre-master key *pmk* according to the chosen scheme, and compute the master secret key *msk* (at this step, we use a random oracle).

The core key-exchange protocol results in a pre-master secret, but the employed mechanisms are too weak to achieve a meaningful security guarantee for this step alone. Obtaining the master secret key *msk* by querying the random oracle "de-correlates" the keys obtained in different sessions and also normalizes their distribution.

The complete step constructs a resource MSK that outputs uniform random keys (the *msk*) to the client and the server; the keys may, however, be different in case the adversary interfered.

4. Key expansion and generating confirmations. Both client and server compute keys κ_C and κ_S and "finished" messages ξ_C and ξ_S via a pseudo-random function PRF keyed with *msk*. The two finished messages are then encrypted, in a concatenation of all the messages in the protocol, and sent.

Since the transmission of ξ_C and ξ_S in the protocol is protected by Authenticate-then-Encrypt with the same keys as the payload messages, a modularization at the intuitively appealing point *after* performing the confirmation is impossible. Only after this protocol step can we fully separate different "sessions," as the computation of ξ_C and ξ_S still requires data which is correlated between different sessions (e.g., the server certificate).

The behavior of this resource is similar to that of MSK with a different key space, but the resource can now be described as the parallel composition of one (single-session) resource $\overset{KSP,*}{=\ =\!\!\bullet}$ per client.

5. Channel setup and payload transmission. The channel setup consists of sending and verifying the messages ξ_C and ξ_S protected with Authenticate-then-Encrypt with keys κ_C and κ_S, respectively. If the verification succeeds, the client and server exchange payload messages protected by Authenticate-then-Encrypt.

The resulting resource $\twoheadleftarrow \overset{*}{\ } \twoheadrightarrow\!\bullet$ is, for each client, one channel to the server which (at the decision of the adversary) allows for either fully secure client-server communication, or for server-adversary communication. The client notices which case occurs, the server does not.

Constructions achieved by the sub-protocols. We sketch the decomposition in Figure 2. The resources NET (the network), PKI (the PKI), and RO (the random oracle) used by the protocol are depicted on the right hand side, and the "converters" that each capture one part of the protocol are drawn as boxes. The first type of converter rnd generates a random nonce. The sub-protocol consisting of one such converter per client constructs the resource NAME that outputs a *unique* nonce to each client (modulo a collision term, which becomes explicit in the security statement).

The second converter hec uses as resources NAME and the network NET, exchanges nonces with the server via NET, and also transmits "payload" messages from higher-level protocols. Together with the server's converter hes, this constructs the "network with nonces," denoted as SNET. The construction statement can be written as

$$\mathsf{NAME} \overset{\mathsf{hec,hes}}{\Longrightarrow} \mathsf{SNET}.$$

The third converter is then alternatively one of dhc (for static DH), dhec (for ephemeral DH), and rsac (for RSA-PKCS) depending on the cipher suite used in the handshake. All these use the resources PKI and RO, as well as the resource SNET constructed by rnd and $(\mathsf{hec},\mathsf{hes})$, and construct, together with the server's counterpart, a "master secret key" resource later called "MSK".

The statements for the following layers are (here for the static DH scheme)

$$[\mathsf{SNET},\mathsf{PKI},\mathsf{RO}] \overset{\mathsf{dhc,dhs}_{\mathcal{G}}}{\Longrightarrow} \mathsf{MSK}, \qquad \mathsf{MSK} \overset{\mathsf{expc,exps}}{\Longrightarrow} \bigotimes_{C\in\mathcal{C}} \left[\!\left[\overset{KSP,*}{=\ =\!\!\bullet} \right]\!\right]^{(C,S/\eta_C)},$$

$$\text{and} \quad \overset{KSP,*}{=\ =\!\!\bullet} \overset{\mathsf{atec,ates}}{\Longrightarrow} \twoheadleftarrow \overset{*}{\ } \twoheadrightarrow\!\bullet\ .$$

In the above statements, $\overset{KSP,*}{=\!=\!\bullet}$ is a resource which is specified with respect to a single client and the server, and the product together with the brackets signifies that there is an independent such resource corresponding to each client. (More details about the notation are in Section 2.) The sub-protocol (expc, exps) expands the secret key and computes the confirmation messages ξ_C and ξ_S, constructing the resource $\overset{KSP,*}{=\!\bullet}$. Finally, the sub-protocol (atec, ates) uses the confirmation messages and also protects payload messages obtained from higher-level protocols, constructing the unilaterally secure channel denoted as $\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet$.

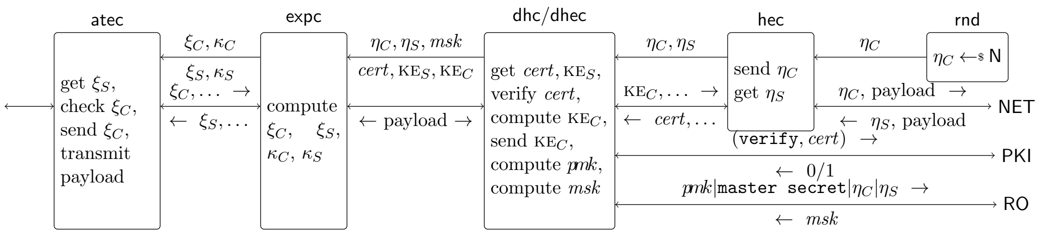
atec
get ξ_S, check ξ_C, send ξ_C, transmit payload
ξ_C, κ_C
ξ_S, κ_S
$\xi_C, \ldots \rightarrow$
$\leftarrow \xi_S, \ldots$
expc
compute ξ_C, ξ_S, κ_C, κ_S
η_C, η_S, msk
$cert$, KE$_S$, KE$_C$
$\leftarrow$ payload $\rightarrow$
dhc/dhec
get $cert$, KE$_S$, verify $cert$, compute KE$_C$, send KE$_C$, compute pmk, compute msk
η_C, η_S
KE$_C, \ldots \rightarrow$
$\leftarrow cert, \ldots$
hec
send η_C get η_S
η_C
rnd
$\eta_C \leftarrow\!\$$ N
η_C, payload $\rightarrow$
$\leftarrow \eta_S$, payload
NET
(verify, $cert$) $\rightarrow$
PKI
$\leftarrow$ 0/1
pmk|master secret|η_C|η_S $\rightarrow$
RO
$\leftarrow msk$

Figure 2: The TLS handshake in terms of client converters

In Figure 2 it becomes apparent that the messages that are intended to be sent over the network and are generated by a higher-level converter are passed through all lower-level converters. Thus, all resources that we specify in the course of the analysis will also have to allow, in addition to their main task, for the insecure transmission of messages. The reason for this is that in the TLS protocol, "TLS fragments" (the messages in TLS are called fragments) are sent over a TCP connection and processed in the order of their arrival. This feature introduces (time-)dependencies between the resources; thus, they cannot be written as a parallel composition of independent resources.

Our results on TLS can be summarized in the following theorem.

Theorem 1 (informal)**.** *Let $\mathcal{C}$ be a set of clients. The TLS protocol constructs, for each client $C \in \mathcal{C}$, one unilaterally secure channel $\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet$ from* NET*,* PKI*, and* RO*, for the static Diffie-Hellman, the ephemeral Diffie-Hellman, and the RSA mode. In more detail:*

1. *The (static) DH-based protocol achieves the construction under the following assumptions: the GapDH assumption holds in the respective group, SHA-256 is collision-resistant, HMAC is a PRF, and the symmetric encryption and MAC schemes are secure.*
2. *The (ephemeral) DH-based protocol achieves the construction under the following assumptions: the GapDH assumption holds in the respective group, the signature scheme used by the server is unforgeable, SHA-256 is collision-resistant, HMAC is a PRF, and the symmetric encryption and MAC schemes are secure.*
3. *The RSA-based protocol achieves the construction under the following assumptions: RSA-PKCS#7 achieves the NR-PCA notion, SHA-256 is collision-resistant, HMAC is a PRF, and the symmetric encryption and MAC schemes are secure.*

A summary of the steps we take and the relevant security loss at each step is given in Figure 11 in the appendix. Note that we only include the details related to the security requirement in the definition of construction, not to the availability requirement.

2 Preliminaries and Notation

2.1 Notation

For a number $n \in \mathbb{N}$, we use the notation $[n] \coloneqq \{1, \ldots, n\}$. Also, for a distribution X over a set $\mathcal{X}$, we write $X \twoheadleftarrow\!\!\$\, \mathsf{X}$ to denote that the random variable X is sampled from distribution X, and we write $X \twoheadleftarrow\!\!\$\, \mathcal{X}$ to express that X is sampled uniformly at random from the set $\mathcal{X}$.

2.2 Constructive Cryptography

The foundational idea of constructive cryptography [MR11, Mau11] is to specify both the assumptions[1] and the guarantees of protocols explicitly as resources, and to consider a protocol as a construction of a (desired) resource from assumed resources. A resource is a shared functionality accessed by several parties; in this work we consider different types of communication channels and shared keys. The assumed resources formalize the setting in which a protocol is used (such as a certain type of communication channel) and constructed resources describe the functionality achieved by using the protocol on the assumed resources (such as a shared key or a communication channel with stronger guarantees). A protocol in this setting is described as a tuple of so-called converters; one per (honest) party.

2.3 Abstract Systems

Resources and converters are modeled as systems. At the highest level of abstraction (following the hierarchy in [MR11]), *systems* are objects with *interfaces* by which they connect to (interfaces of) other systems; each interface is labeled with an element of some label set and connects to only a single other interface. Multiple interfaces can be merged into a single interface; the original interfaces are referred to as *sub-interfaces* of the composite interface. (Sub-interfaces have labels relative to the composite interface; to refer to a sub-interface S of some interface I, we write I/S.) This concept of *abstract systems* captures the topological structures that result when multiple systems are connected in this manner. It does not, however, model the behavior of systems, i.e., *how* the systems interact via their interfaces; statements about specific protocols are statements at this next (lower) abstraction level. In this work, we describe all systems in terms of (probabilistic) discrete systems.

Resources and converters. *Resources* in this work are systems with interfaces labeled by elements of some label set $\mathcal{L}$. A *converter* is a two-interface system which is directed in that it has an *inside* and an *outside* interface. As a notational convention, we generally use upper-case, bold-face letters (e.g., $\mathbf{R}$, $\mathbf{S}$), symbols (e.g., $\bullet\!\!-$ $\rightarrow$), or upper-case sans-serif fonts to denote resources, and lower-case Greek letters (e.g., α, β) or sans-serif fonts (e.g., enc, dec) for converters. We denote by $\Phi_{\mathcal{L}}$ (or simply Φ if $\mathcal{L}$ is clear from the context) the set of all resources with interface labels in $\mathcal{L}$, and by Σ the set of all converters. We use two special converters, an "identity" converter id and a "blocking" converter $\perp$ that has an inactive outside interface.

System composition. The topology of a composite system is described using a term algebra, where each expression starts from one resource on the right-hand side and is subsequently extended with further terms on the left-hand side. An expression is interpreted in the way that all interfaces of the system it describes can be connected to interfaces of systems which are appended on the left. For instance, for a single resource $\mathbf{R} \in \Phi$, all its interfaces are accessible.

[1]The term "assumption" often refers to two different concepts: setup assumptions such as a network or a PKI, and computational assumptions such as the hardness of certain problems. Here, we refer to setup assumptions.

For $I \in \mathcal{L}$, a resource $\mathbf{R} \in \Phi$, and a converter $\alpha \in \Sigma$, the expression $\alpha^I \mathbf{R}$ denotes the composite system obtained by connecting the inside interface of α to the I-interface of $\mathbf{R}$; the outside interface of α becomes the I-interface of the composite system. The system $\alpha^I \mathbf{R}$ is again a resource. For two resources $\mathbf{R}$ and $\mathbf{S}$, $[\mathbf{R}, \mathbf{S}]$ denotes the parallel composition of $\mathbf{R}$ and $\mathbf{S}$. For each $I \in \mathcal{L}$, the I-interfaces of $\mathbf{R}$ and $\mathbf{S}$ are merged and become the *sub-interfaces* of the I-interface of $[\mathbf{R}, \mathbf{S}]$. A converter α that connects to the I-interface of $[\mathbf{R}, \mathbf{S}]$ has two inside sub-interfaces, where the first connects to $\mathbf{R}$ and the second connects to $\mathbf{S}$ (i.e., sub-interfaces are ordered). Finally, we denote by id a special converter that forwards all messages from the inside to the outside interface and vice versa, hence $\mathsf{id}^I \mathbf{R} \equiv \mathbf{R}$.

We introduce special notation for families of resources or converters: If we compose a family of resources $(\mathbf{R}_i)_{i \in \{1,\dots,n\}}$ (resp. converters $(\alpha_i)_{i \in \{1,\dots,n\}}$) in parallel, we write this as a product such as $\bigotimes_{i=1}^n \mathbf{R}_n$ (resp. $\bigotimes_{i=1}^n \alpha_i$). If we attach a family of converters $\alpha_1, \dots, \alpha_n$ to interfaces $I_1, \dots, I_n$ of a resource $\mathbf{R}$, we write $\prod_{i=1}^n {\alpha_i}^{I_i} \mathbf{R}$.

Each setting is described by a cryptographic algebra with a certain label set $\mathcal{L}$.

Definition 2. A *cryptographic algebra* for a label set $\mathcal{L}$ is a pair $\langle \Phi, \Sigma \rangle$ consisting of a set of resources Φ and a set of converters Σ, together with families of parallel composition operations $[\cdot] : \Phi^* \to \Phi$ and $[\cdot] : \Sigma^* \to \Sigma$, as well as connecting operations $\cdot^i \cdot : \Sigma \times \Phi \to \Phi$. The algebra is *composition-order independent* if

1. for all $C_1, C_2 \in \Sigma$, $R \in \Phi$, and $i, j \in \mathcal{L}$ with $i \neq j$,
$$C_1^i (C_2^j R) = C_2^j (C_1^i R), \text{ and}$$
2. for all $C_1, \dots, C_m \in \Sigma$, $R_1, \dots, R_m \in \Phi$, and $i \in \mathcal{L}$,
$$[C_J]^i [R_J] = \left[C_1^i R_1, \dots, C_m^i R_m\right].$$

Distinguishers. A *distinguisher* $\mathbf{D}$ is a special type of system that connects to all interfaces of a resource $\mathbf{U}$ and outputs a single bit at the end of its interaction with $\mathbf{U}$. In the term algebra, this appears as the expression $\mathbf{DU}$, which defines a binary random variable. The *distinguishing advantage of a distinguisher* $\mathbf{D}$ *on two systems* $\mathbf{U}$ *and* $\mathbf{V}$ is defined as

$$\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) \coloneqq |\Pr(\mathbf{DU} = 1) - \Pr(\mathbf{DV} = 1)|.$$

The advantage of a class $\mathcal{D}$ of distinguishers is defined as $\Delta^{\mathcal{D}}(\mathbf{U}, \mathbf{V}) \coloneqq \sup_{\mathbf{D} \in \mathcal{D}} \Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V})$. The distinguishing advantage measures how much the distribution of the output of $\mathbf{D}$ differs when it is connected to either $\mathbf{U}$ or $\mathbf{V}$. Intuitively, if no distinguisher (of a certain class) differentiates between $\mathbf{U}$ and $\mathbf{V}$, they can be used interchangeably in any environment of that class (otherwise that specific environment could serve as a distinguisher).

Note that the distinguishing advantage is a pseudo-metric. In particular, it satisfies the triangle inequality, i.e., $\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{W}) \leq \Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) + \Delta^{\mathbf{D}}(\mathbf{V}, \mathbf{W})$ for all resources $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ and distinguishers $\mathbf{D}$. There is an *equivalence* relation on the set of resources (which is defined on the level of discrete systems), denoted by $\mathbf{U} \equiv \mathbf{V}$, which means that $\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) = 0$ for *all* distinguishers $\mathbf{D}$.

Games. Games that capture properties such as unforgeability are two-interface systems that at their left interface connect to some adversary or solver $\mathbf{A}$ and at the right interface output a single bit (usually denoted W). The performance of $\mathbf{A}$ in a game $\mathbf{G}$ is denoted as

$$\Gamma^{\mathbf{A}}(\mathbf{G}) \coloneqq \Pr^{\mathbf{AG}}(W = 1).$$

Reductions. When relating (the hardness of) two problems such as distinguishing systems or winning a game, it is convenient to use a special type of system $\mathbf{C}$ that translates one setting into the other. Formally, $\mathbf{C}$ is a converter that has an *inside* and an *outside* interface. When it is connected to a system $\mathbf{S}$, which is denoted by $\mathbf{CS}$, the inside interface of $\mathbf{C}$ connects to the merged interfaces of $\mathbf{S}$ and the outside interface of $\mathbf{C}$ becomes the interface of the composed system. $\mathbf{C}$ is called a *reduction system* (or simply *reduction*).

To reduce distinguishing two systems $\mathbf{S}, \mathbf{T}$ to distinguishing two systems $\mathbf{U}, \mathbf{V}$, one describes a reduction $\mathbf{C}$ such that $\mathbf{CS} \equiv \mathbf{U}$ and $\mathbf{CT} \equiv \mathbf{V}$. Then, for all distinguishers $\mathbf{D}$, we have $\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) = \Delta^{\mathbf{D}}(\mathbf{CS}, \mathbf{CT}) = \Delta^{\mathbf{DC}}(\mathbf{S}, \mathbf{T})$. The last equality follows from the fact that $\mathbf{C}$ can also be thought of as being part of the distinguisher.

2.4 The Notion of Construction

The *construction* notion (cf. [Mau11]) requires two conditions: *availability* and *security.* For the former, the behavior of the real resource (with the converters) must be indistinguishable to the behavior of the ideal resource, if the "blocking" converter $\perp$ is attached at the adversarial interface (always denoted by the special label E) of both resources. For the latter requirement, we demand the existence of a simulator which essentially emulates the behavior at the E-interface of the real resource, while being connected to the corresponding interface of the ideal resource. More formally:

Definition 3. Let $\Phi_{\mathcal{L}}$ and Σ be as above. A protocol $\pi_{\mathcal{L}'} = (\pi_\ell)_{\ell \in \mathcal{L}'}$ *constructs resource* $\mathbf{S} \in \Phi$ *from resource* $\mathbf{R} \in \Phi$ *within* ε *and with respect to distinguisher class* $\mathcal{D}$, if

$$\begin{cases} \qquad \Delta^{\mathcal{D}}\left((\pi_{\mathcal{L}'})^{\mathcal{L}'} {\pi_S}^S \perp^E \mathbf{R}, \perp^E \mathbf{S}\right) \leq \varepsilon & (\textit{availability}) \\ \exists \sigma \in \Sigma : \quad \Delta^{\mathcal{D}}\left((\pi_{\mathcal{L}'})^{\mathcal{L}'} {\pi_S}^S \mathbf{R}, \sigma^E \mathbf{S}\right) \leq \varepsilon & (\textit{security}), \end{cases}$$

where $(\pi_{\mathcal{L}'})^{\mathcal{L}'}$ means attaching each π_ℓ to interface ℓ.

In the descriptions of the resources, we use a special "cheating bit" $b \in \{0, 1\}$ at the E-interface and describe their behavior in the case there is *no* attacker present ($b = 0$, this is input by $\perp$), and in case that there *is* an attacker present ($b = 1$, usually set by the simulator).

Settings and interface sets considered in this work. The most important scenario we consider in this work comprises multiple clients, one server, and one (explicit, external) adversary. Some resources are easier described as the parallel composition of resources for fewer parties and some protocol steps can be proven in such a simpler setting in isolation, i.e., with respect to only one client, one server, and the adversary.

The simplified setting involving only two honest parties and one attacker is called the $\{A, B, E\}$*-setting* and is used to analyze protocols and schemes like symmetric encryption or MACs, cf. [MT10]. The (honest) parties' interfaces are named A and B, and there is an explicit adversarial interface E. Resources are in the set $\Phi_{\{A,B,E\}}$, and protocols are pairs of converters $\pi = (\pi_1, \pi_2)$ for A and B, respectively.

Unilateral key-exchange protocols are used in a setting with multiple clients, one server, and an explicit adversary. We consider a set $\mathcal{C}$ of clients, a server S, and an adversary E. Hence, we consider a label set $\mathcal{L} = \mathcal{C} \cup \{E, S\}$, resources are in the set $\Phi_{\mathcal{L}}$, and a protocol consists of a family $(\pi_C)_{C \in \mathcal{C}}$ of client converters and a server converter π_S.

Constructions in the $\{A, B, E\}$-setting can be "lifted" to settings with more interfaces. Such a lifting is described by an injective function $\tau : \{A, B, E\} \to \mathcal{L}$, where we generally assume $\tau(E) = E$. Resources $\mathbf{R} \in \Phi_{\{A,B,E\}}$ are embedded into $\Phi_{\mathcal{L}}$ by providing the A and B-interfaces as $\tau(A)$ and $\tau(B)$-interfaces and inactive interfaces for all $I \in \mathcal{L} \setminus \tau(\{A, B, E\})$. We denote this resource by $[\![\mathbf{R}]\!]^{(\tau(A),\tau(B),\tau(E))}$ (we usually only write $[\![\mathbf{R}]\!]^{(\tau(A),\tau(B))}$). A protocol $\pi = (\pi_1, \pi_2)$ consisting of a pair of converters π_1 for A and π_2 for B becomes $\pi_{\mathcal{L}} = (\pi_I)_{I \in \mathcal{L}}$ with $\pi_{\tau(A)} = \pi_1$, $\pi_{\tau(B)} = \pi_2$, and $\pi_I = \mathsf{id}$ for all $I \notin \tau(\{A, B, E\})$. Security statements transfer from the $\{A, B, E\}$-setting to the $\mathcal{L}$-setting since any distinguisher in the $\mathcal{L}$-setting can be translated into a distinguisher for the $\{A, B, E\}$-setting by simply emulating the inactive interfaces. (In particular this is a mapping in the above described sense.)

2.5 Discrete Systems

The statements in this paper are statements about *discrete systems*, i.e., systems that communicate by receiving and sending messages. We formalize these systems as random systems [Mau02, Mau13], i.e., families of conditional probability distributions.

Definition 4. Let $\mathcal{X}$ and $\mathcal{Y}$ be sets. An $(\mathcal{X}, \mathcal{Y})$*-random system* $\mathbf{F}$ is an infinite sequence of conditional probability distritbutions $\mathsf{p}^{\mathbf{F}}_{Y_i|X^iY^{i-1}} : \mathcal{Y} \times \mathcal{X}^i \times \mathcal{Y}^{i-1} \to [0, 1]$ for $i \geq 1$.

Random systems can be equivalently described by a sequence of conditional probability distributions $\mathsf{p}^{\mathbf{F}}_{Y^i|X^i}$ for $i \geq 1$ that satisfy a compatibility condition, i.e., for any two $i' < i''$ the distributions are consistent for the first i' values.

A game as described in Section 2.3 is an $(\mathcal{X}, \mathcal{Y})$-system which interacts with its environment by taking inputs $X_1, X_2, \ldots$ (considered as moves of the adversary) and answering with outputs $Y_1, Y_2, \ldots$. In addition, after every input it also outputs a bit indicating whether the game has been won. This bit is monotone in the sense that it is initially set to 0 and that, once it has turned to 1 (the game is won), it can not turn back to 0 (even if the game were continued). This motivates the following definition, which captures the notion of game-winning.

Definition 5. For a $(\mathcal{X}, \mathcal{Y} \times \{0, 1\})$-system $\mathbf{F}$ the binary component A_i of the output (Y_i, A_i) is called a *monotone binary output (MBO)* if $A_i = 1$ implies $A_j = 1$ for $j \geq i$. Such a system $\mathbf{F}$ with MBO is also called a *(discrete) game*.

Any system $\mathbf{F}$ together with a *monotone event sequence (MES)* $\mathcal{A} = (A_1, A_2, \ldots)$ defined on $\mathbf{F}$ (within any random experimant, see [Mau02]) can be seen as the game by extending the output of $\mathbf{F}$ by an additional component which turns from 0 to 1 exactly when the MES becomes false. We write $\mathbf{F}^{\mathcal{A}}$ to denote the system with the additional MBO. Conversely, for a $(\mathcal{X}, \mathcal{Y} \times \{0, 1\})$-system $\mathbf{F}$ (with MBO) we write $\mathbf{F}^{-}$ to denote the $(\mathcal{X}, \mathcal{Y})$ system where the binary component of the output is discarded.

The adversary (or game winner) and the game are connected via their interfaces; i.e., the adversary specifies the inputs X_i and obtains the outputs Y_i of the game. To formulate "traditional" game-based definitions in this language, the game, often denoted as $\mathbf{G}$ with additional super- and subscripts, allows the adversary $\mathbf{A}$ to issue "oracle queries" via that interface. We usually denote the special monotone output bit as W. For a game $\mathbf{G}$ and an adversary $\mathbf{A}$, we define the *game-winning probability* after q steps (queries) as

$$\Gamma^{\mathbf{A}}_q(\mathbf{G}) \coloneqq \mathbb{P}^{\mathbf{AG}}(W_q = 1).$$

For an adversary $\mathbf{A}$ that halts after (at most) q steps, we write $\Gamma^{\mathbf{A}}(\mathbf{G}) \coloneqq \Gamma^{\mathbf{A}}_q(\mathbf{G})$.

Two systems with MBOs can be *equivalent as games*, which captures only that they behave equivalently as long as the game is not won.

Definition 6. Two $(\mathcal{X}, \mathcal{Y} \times \{0, 1\})$-systems with MBO, $\mathbf{S}$ and $\mathbf{T}$, are *equivalent as games*, denoted $\mathbf{S} \overset{g}{\equiv} \mathbf{T}$, if, for $i \geq 1$,

$$\mathsf{p}^{\mathbf{S}}_{Y^i, A_i=0|X^i} = \mathsf{p}^{\mathbf{T}}_{Y^i, A_i=0|X^i}.$$

2.6 Insecure Communication Channels and TLS Fragments

The TLS protocol transmits its data via a TCP connection which transmits a byte stream. At the lowest level, the TLS record protocol then partitions this byte stream into *fragments*, each fragment corresponding to one message sent in the TLS protocol. Technically, each fragment consists of four fields:

1. the *content type* of the transmitted message, which signals whether the message is a `change_cipher_spec` request, an `alert`, a `handshake` message, or `application_data`,
2. the *protocol version*,
3. the *length* of the following payload, and
4. the *payload* message itself.

For simplicity, we model the communication channels that we assume as channels that transmit TLS fragments, ignoring the part of the protocol that converts the fragments into the TCP byte stream and vice versa. The plaintext contained in a TLS plaintext fragment is of length at most $2^{14} = 16384$ bits, and a TLS ciphertext fragment is of length at most $2^{14} + 2048 = 18432$ bits (plus 5 bytes of header). We generally write $\{0,1\}^{\leq \ell} \coloneqq \bigcup_{0 \leq i \leq \ell} \{0,1\}^i$.

3 Constructing the Master Secret

3.1 The Assumed Resources

The resources we assume for the TLS protocol are: an insecure network (obtained by using the TCP/IP protocol in the Internet), a public-key infrastructure which we view as allowing the server to send one message (its public key) authentically to all clients, and a random-oracle resource, outputting consistent random values for user (and adversary) input.

The "Internet" network resource. The insecure network is described as a parallel composition of insecure channels. There are two such channels between each potential client and the server (one in each direction). Technically, the TLS protocol is run via a TCP connection which transmits byte streams; TLS partitions such a byte stream into *TLS fragments* that correspond to the messages sent by higher protocol layers. These fragments are the lowest layer that we consider in our analysis; for us, an insecure communication channel is one that transmits a sequence of TLS fragments.

Insecure channel $- \twoheadrightarrow$

0. Obtain input $b \in \{0,1\}$ at the E-interface.
1. (Repeatedly) Upon input a message $m \in \{0,1\}^{\leq 18432}$ at the A-interface:
   - if $b = 0$, then output m at the B-interface;
   - if $b = 1$, then output m at the E-interface.
2. (Repeatedly) Upon input a message $m \in \{0,1\}^{\leq 18432}$ at the E-interface, if $b = 1$, then output m at the B-interface.

As TLS builds on TCP, we can envision the insecure channels to be identified by the clients' TCP sockets, i.e., an address in the set $\mathcal{A}_{\mathrm{TCP}} = \{0,1\}^{32} \times \{0,1\}^{16}$. Consequently, the entire network resource, which we denote by NET, is defined as the parallel composition of resources $\bigotimes_{C \in \mathcal{A}_{\mathrm{TCP}}} [\![- \twoheadrightarrow, \twoheadleftarrow -]\!]^{(C,S/C)}$.

The Public-Key Infrastructure. For the case of unilateral authentication (i.e., only the server has a certificate and the clients are not authenticated), we describe a "simple" resource that allows the server to authenticate its public key toward all clients. This task is, in a real protocol, achieved by using certificates obtained from some certification authority.

The resource PKI provides one "client" interface for each $C \in \mathcal{C}$, as well as a "server" interface S, and an "eavesdropper" interface E. The resource is parametrized by a distribution $\mathfrak{F}$ over functions $f : \{0,1\}^* \to \{0,1\}^*$ that depends on the scheme used to construct the resource, which might be, as in the case of TLS, X.509. For more details on X.509 see Appendix A.1.

Public-Key Infrastructure Resource $\mathsf{PKI}_{\mathfrak{F}}$

Choose a function $f \leftarrow \mathfrak{F}$.

- On input a message $(\texttt{register}, x)$ at the S-interface with $x \in \{0,1\}^*$, output $s = f(x)$ at the S-interface. Additionally output (x, s) at the E-interface.
- On input $(\texttt{verify}, x, s)$ at some interface $C \in \mathcal{C}$:
  - If x was input at the S interface and the response was s, then output 1.
  - Otherwise, output 0.

Related papers following the constructive cryptography paradigm, such as [CMT13, MTC13], model the capability of authentically transmitting a single message such as a public key to other parties as an authenticated single-message channel. This simpler formalization is generally preferable because it abstracts from the scheme with which the authentication is achieved; however, in the case of TLS the certificates issued by the certification authority (i.e., the bit strings) are actually used within higher levels of the protocol, in particular, the certificates are included in the computation of the "finished" message.

Uniqueness of certificates. The description of the certification resource $\mathsf{PKI}_{\mathfrak{F}}$ described above formalizes that the encoding of certificates is unique: the verification is successful if and only if *exactly the same* pair (x, s) has occurred previously. This assumption is justified because the X.509 certificates are transmitted in TLS via DER encoding, which guarantees that every data structure has a unique representation.

The random oracle. The random oracle is a resource that can be queried at all interfaces with strings $x \in \{0,1\}^*$, and for each new string, it responds with a uniform output of length k. If some input was queried (at any interface) before, then the random oracle answers consistently.

Random Oracle RO_k

On input a string $x \in \{0,1\}^*$ at some interface:

- if x was queried before at some (potentially other) interface, respond with the same value as in that query;
- otherwise, draw a uniformly random $y \in \{0,1\}^k$ and respond with y.

3.2 Session Naming

In the setting of unilateral key exchange, clients do not *a priori* possess any information that differentiates them from other clients. To prevent protocol sessions from interfering with each other, however, it is necessary to cryptographically bind some unique information to each session. In the TLS protocol, the client's nonce can serve this purpose. In fact, this nonce can be seen as a "unique name", as it is—with high probability—unique for honest clients.

3.2.1 Choosing Unique Nonces

Unique names are formalized as a resource that provides to each honest party a unique value. A unique name in this sense is a weak assumption, as there is no authenticity requirement. For a set $\mathcal{N}$ of names and a set $\mathcal{C} \subseteq \mathcal{A}_{\mathrm{TCP}}$ such that $|\mathcal{N}| \geq |\mathcal{C}|$, the resource NAME parametrized by an injective function $\rho : \mathcal{C} \to \rho(\mathcal{C}) \subseteq \mathcal{N}$ assigns to each client $C \in \mathcal{C}$ a unique name $\eta = \rho(C) \in \mathcal{N}$.

Unique name resource NAME_ρ for $\rho : \mathcal{C} \to \mathcal{N}$

At each interface $C \in \mathcal{C}$, output $\rho(C)$.

Choosing a nonce at random from some distribution N over the set $\mathcal{N}$ also implements this resource; thus, the resource is constructed without any setup assumptions. In more detail, let rnd be the converter that chooses a nonce $\eta \leftarrow\!\!\$\ \mathsf{N}$ at random and outputs η at the outside interface. The client nonces in TLS consist of a 28 byte random string to which the date and time are prepended, so $\mathcal{N} = \{0,1\}^{256}$. As the nonce contains 224 bits of randomness, any pair of two clients chooses the same nonce with probability at most $\binom{|\mathcal{C}|}{2}2^{-224}$, where $|\mathcal{C}|$ is the total number of client sessions (see [MTC13, Lemma 7]). In the following, we use the symbol N to denote the distribution used in TLS.

3.2.2 Separating Network Sessions

The client's nonce η_C is sent to the server which uses it to identify that particular client's session. While the nonces used by honest clients as obtained from the NAME resource are unique, the server does not prevent the adversary from starting many sessions with the same nonce. Hence, we index the sessions by pairs $sid = (\eta_C, e) \in \mathcal{N} \times \mathbb{N}$. Furthermore, the server also chooses a nonce $\eta_{sid} \leftarrow\!\!\$\ \mathsf{N}$ for this session and sends it to the client. This nonce exchange protocol constructs, from the resources NAME_ρ for some injective mapping $\rho : \mathcal{C} \to \mathcal{N}$ and NET, a resource denoted by SNET.

The resource SNET, wich is described in Figure 3, has interfaces labeled $C \in \mathcal{C}$ for the clients, a server's interface S which has one sub-interface for each pair (η, e), where $\eta \in \mathcal{N}$ is the client's nonce and $e \in [n]$ is a counter indicating how many sessions have been initiated with nonce η, and an adversary's interface called E. To simplify further construction steps, we rule out collisions for *server* nonces in the SNET resource below, in sessions that are associated to the same client nonce (i.e., $sid = (\eta_C, e)$ and $sid' = (\eta_C, e')$). The reason is that by making the pairs of clients and server nonces unique, we guarantee that any two sessions obtain their keys by querying different parts of the random oracle. Technically, our SNET resource will check, when generating a server nonce for a particular client nonce, whether the same server nonce has already been generated for that client nonce. Note that the server nonce has the same structure as the client nonce, thus the security loss is analogous.

The resource $\mathsf{SNET}_{\mathsf{N},\rho,n}$ is constructed (from NAME_ρ and NET) by the protocol $(\mathsf{hec}, \mathsf{hes}_n)$ described as follows, where the server's converter hes_n is parametrized by the maximum number of sessions it accepts per client nonce.

Client converter hec: (Each such converter connects to an interface $C \in \mathcal{C} \subseteq \mathcal{A}_{\mathrm{TCP}}$ which has three sub-interfaces: one each for sending messages to, and respectively receiving messages from the server, and a third sub-interface for obtaining the nonce.) Get a nonce $\eta_C \in \mathcal{N}$ from the resource NAME_ρ and send it via the sending sub-interface. Upon receiving $\tilde{\eta}$ at the receiving sub-interface, output $(\eta_C, \tilde{\eta})$ at the outside interface. Afterward, forward all messages (between the sending and receiving sub-interfaces).

Server converter hes_n: Initially set $e_\eta = 1$ for each $\eta \in \mathcal{N}$.

Augmented Network Resource $\mathsf{SNET}_{\mathsf{N},\rho,n}$

The resource is parametrized by a nonce distribution N, an injective function $\rho : \mathcal{C} \to \mathcal{N}$, and $n \in \mathbb{N}$.

Obtain at the E-interface an input $b \in \{0,1\}$. If $b = 0$, then for each $C \in \mathcal{C}$ choose a nonce $\eta_{sid} \leftarrow\$ \mathsf{N}$ with $sid = (\rho(C), 1)$, output $(\rho(C), \eta_{sid})$ at interface C and η_{sid} at interface sid, and provide bidirectional channels between those two interfaces.

Otherwise, if $b = 1$, then initialize $e_\eta = 0$ for all $\eta \in \mathcal{N}$ and

- Upon input $(\mathtt{ack}, \tilde{\eta})$ at interface E, if $e_{\tilde{\eta}} \leq n$, then for $sid = (\tilde{\eta}, e_{\tilde{\eta}})$ choose a nonce η_{sid} uniformly at random from $\mathcal{N} \setminus \{\eta_{sid_1}, \ldots, \eta_{sid_{e_{\tilde{\eta}}-1}}\}$ (where $sid_e = (\tilde{\eta}, e)$), output η_{sid} at the E/sid- and the S/sid-interfaces, and increase $e_{\tilde{\eta}}$. Then, forward all communication between the interfaces S/sid and E/sid.
- Upon input $(\mathtt{deliver}, C, \tilde{\eta})$ with $C \in \mathcal{C}$, output $(\eta_C, \tilde{\eta})$ at the interface C. Then, forward all communication between the interfaces C and E/C.

Figure 3: The network resource that additionally outputs nonces.

- Upon receiving a nonce $\tilde{\eta}_C$ at an inside C-sub-interface for some $C \in \mathcal{A}_{\mathrm{TCP}}$, if $e_{\tilde{\eta}_C} \leq n$, then choose a nonce $\eta_{sid} \leftarrow\$ \mathsf{N}$, send η_{sid} via the C-sub-interface at the inside. Output η_{sid} at outside $\tilde{sid}$-sub-interface with $\tilde{sid} = (\tilde{\eta}_C, e_{\tilde{\eta}_C})$, and increment $e_{\tilde{\eta}_C}$ by 1.
- Afterward, forward all messages between the outside sub-interfaces associated to $\tilde{sid}$ and the inside C-sub-interfaces.

We prove the following (security) statement.

Lemma 7. *Let $\mathcal{C} \subseteq \mathcal{A}_{\mathrm{TCP}}$ and let $\rho : \mathcal{C} \to \mathcal{N}$ be an injective mapping. The protocol $(\mathsf{hec}, \mathsf{hes}_n)$ constructs the resource $\mathsf{SNET}_{\mathsf{N},\rho,n}$ from the resources NET and NAME_ρ. More concretely,*

$$\prod_{C \in \mathcal{C}} \mathsf{hec}^C \mathsf{hes}_n{}^S \bot^E [\mathsf{NET}, \mathsf{NAME}_\rho] \quad \equiv \quad \bot^E \mathsf{SNET}_{\mathsf{N},\rho,n},$$

and there exists a simulator σ such that, for all distinguishers $\mathbf{D}$,

$$\Delta^{\mathbf{D}} \left(\prod_{C \in \mathcal{C}} \mathsf{hec}^C \mathsf{hes}_n{}^S [\mathsf{NET}, \mathsf{NAME}_\rho], \sigma^E \ \mathsf{SNET}_{\mathsf{N},\rho,n} \right) \leq \binom{n}{2} \cdot 2^{-224}.$$

Proof.

Availability. Apart from outputting nonces at the server's side, there is one main difference between the real resource, $[\mathsf{NET}, \mathsf{NAME}_\rho]$, and the ideal resource, $\mathsf{SNET}_{\mathsf{N},\rho,n}$: the server sessions in $\mathsf{SNET}_{\mathsf{N},\rho,n}$ are denoted by tuples (η, e), where η is the client nonce and $e \in [n]$ is the index of the session between the server and that particular client; by contrast, the server sessions in NET are denoted as TCP addresses. Since η is unique (by the properties of NET), this difference is strictly nominal, and the server's converter hes_n takes care of it.

Security. We prove the security statement in two steps. In the first step we modify the ideal resource $\mathsf{SNET}_{\mathsf{N},\rho,n}$ and define a resource $\mathbf{H}$ which behaves exactly like $\mathsf{SNET}_{\mathsf{N},\rho,n}$ except that it does not check for collisions in server nonce values. We first argue that $\mathsf{SNET}_{\mathsf{N},\rho,n}$ and $\mathbf{H}$ behave identically, up to a difference of $\binom{n}{2} \cdot 2^{-224}$. Indeed, the two resources behave differently only when, for some nonce $\tilde{\eta}$, the resource $\mathbf{H}$ outputs a duplicate nonce $\eta_{sid} = \eta_{sid'}$ for $sid = (\tilde{\eta}, e) \neq (\tilde{\eta}, e') = sid'$. A collision in two of at most n sessions occurs with probability $\binom{n}{2} \cdot 2^{-224}$.

Let $\mathbf{R} \coloneqq \prod_{C \in \mathcal{C}} \mathsf{hec}^C \mathsf{hes}_n{}^S \bot^E [\mathsf{NET}, \mathsf{NAME}_\rho]$. In the second step, we show $\mathbf{R}$ and $\mathbf{S}' \coloneqq \sigma^E \mathbf{H}$ are equivalent, for the following σ described below. (Note that intuitively $\mathbf{S}'$ (and indeed the ideal resource $\mathsf{SNET}_{\mathsf{N},\rho,n}$) has no security properties with respect to the message transmission, e.g. integrity or confidentiality. Indeed, $\mathsf{SNET}_{\mathsf{N},\rho,n}$ describes a network that exchanges nonces correctly between clients and servers, with the additional property that no two sessions share the same nonces.) The simulator σ behaves as follows:

- **Initialization.** Set $e_\eta = 1$ for all $\eta \in \mathcal{N}$. For each $C \in \mathcal{C}$, simulate $\rho(C)$ as being transmitted on $[\![- \twoheadrightarrow]\!]^{(C,S/C)}$.

- **Delivery of client's nonce.** On input a nonce $\tilde{\eta}$ at the outside sub-interface corresponding to a channel $[\![- \twoheadrightarrow]\!]^{(C,S/C)}$, if $e_{\tilde{\eta}} \leq n$, then input $(\mathtt{ack}, \tilde{\eta})$ at the inside interface, set $sid = (\tilde{\eta}, e_{\tilde{\eta}})$, and increase $e_{\tilde{\eta}}$. Obtain a nonce at the inside interface, call it η_{sid}, and output η_{sid} at the outside interface as transmitted via $[\![\twoheadleftarrow -]\!]^{(C,S/C)}$. All messages obtained via the inside sid-sub-interface are output as transmitted via $[\![\twoheadleftarrow -]\!]^{(C,S/C)}$, all messages input at the outside sub-interface corresponding to channel $[\![- \twoheadrightarrow]\!]^{(C,S/C)}$ are input at the inside sid-sub-interface.

- **Delivery of the server's nonce.** On input the first message (a nonce $\tilde{\eta}$) at the outside sub-interface corresponding to a channel $[\![\twoheadleftarrow -]\!]^{(C,S/C)}$, input $(\mathtt{deliver}, C, \tilde{\eta})$ at the inside interface. All messages obtained via the inside C-sub-interface are output as transmitted via $[\![- \twoheadrightarrow]\!]^{(C,S/C)}$, all messages input at the outside sub-interface corresponding to channel $[\![\twoheadleftarrow -]\!]^{(C,S/C)}$ are input at the inside C-sub-interface.

We argue that the simulation is perfect. Firstly, the simulator correctly instantiates the sessions between clients and server. Indeed, in both cases, consistent server sub-interfaces are used because the counters e_{η_C} are updated whenever a nonce η_C is delivered to the server.

The equivalence then follows from two observations:

- On input the first message, a nonce $\tilde{\eta}$, at the E-interface of some channel $[\![- \twoheadrightarrow]\!]^{(C,S/C)}$, both the real and the ideal systems will respond with a nonce $\eta_{sid} \leftarrow\$ \mathsf{N}$ for $sid = (\tilde{\eta}, e_{\tilde{\eta}})$, and also in both cases the output at the interface S/sid is (η_C, η_{sid}). Afterward, the communication between the interfaces S/sid and the E-interfaces of the channels corresponding to C' is forwarded in both cases.
- On input the first message, a nonce $\tilde{\eta}$, at the E-interface of some channel $[\![\twoheadleftarrow -]\!]^{(C,S/C)}$, both the real and the ideal systems output $\tilde{\eta}$ at the C-interface. Afterward, the communication between the interface C and the E-interfaces of the channels corresponding to C is forwarded in both cases.

This concludes the proof. □

3.3 The Master Secret Key Resource

The next step, following the structure of our outline, is constructing the master secret key. In more detail, we construct (from SNET, PKI, and RO) a resource denoted as $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$.

Intuitively, this resource represents the result of the core of the handshake itself, as the parties output the master key for the session, from which other keys are derived.

The resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ modeling the TLS master secret key (described in Figure 4) has interfaces $C \in \mathcal{C} \subseteq \mathcal{A}_{\text{TCP}}$ for the clients, an interface S for the server with sub-interfaces labeled $sid = (\eta, e) \in \mathcal{N} \times [n]$, and an "adversarial" interface E, and is parametrized by the maximum number n of sessions per client nonce. For the set $\mathcal{C} \subseteq \mathcal{A}_{\text{TCP}}$, consider an injective function $\rho : \mathcal{C} \to \mathcal{N}$, which also parametrizes the resource. The resource is further parametrized by a distribution AUX for the auxiliary information—this is important for specifying the behavior of the resource in the availability case. More generally, this can be a family of distributions to formalize that the auxiliary information in different sessions has some correlated random information.

Master Secret Key Resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$

Accept at the E-interface an input $b \in \{0,1\}$. If $b = 0$, then, for each $C \in \mathcal{C}$, choose a key $\kappa_C \leftarrow\$ \{0,1\}^{384}$, a nonce $\eta_{sid} \leftarrow\$ \mathsf{N}$ (with $sid = (\rho(C), 1)$), and auxiliary information $aux_C \leftarrow\$ AUX$, and output $(\kappa_C, \rho(C), \eta_{sid}, aux_C)$ at interface C and $(\kappa_C, \eta_{sid}, aux_C)$ at the interface S/sid. Then, behave as the parallel composition $\bigotimes_{C\in\mathcal{C}} [\![- \twoheadrightarrow, \twoheadleftarrow -]\!]^{(C,S/\rho(C))}$ with the cheating bits $b \in \{0,1\}$ of all channels set to $b = 0$.

Otherwise, if $b = 1$, then: Initialize $e_\eta = 1$ for all $\eta \in \rho(\mathcal{C})$.

- For each $\eta \in \mathcal{N}$, upon input $(\texttt{ack}, \eta)$ at the E-interface, if $e_\eta \leq n$, then choose η_{sid} uniformly at random from $\mathcal{N} \setminus \{\eta_{sid_0}, \dots, \eta_{sid_{e_{\tilde{\eta}}-1}}\}$ for $sid = (\eta, e_\eta)$, output η_{sid} at the E-interface and increase e_η.
- Upon input $(\texttt{allow}, C, aux, \tilde{\eta})$ at the E-interface with $aux \in \{0,1\}^*$, and $\tilde{\eta} \in \mathcal{N}$, output at the C-interface $(\kappa_C, \rho(C), \tilde{\eta}, aux)$ with a uniformly random $\kappa_C \in \mathcal{K}$.

  Afterward, if at the C-interface a message $m \in \{0,1\}^{\leq 18432}$ is input, output m at the E-interface. Also, allow at the E-interface to inject messages $m' \in \{0,1\}^{\leq 18432}$, to be output at the C-interface.
- Upon input $(\texttt{deliver}, \eta, e, aux)$ at the E-interface: if $e < e_\eta$ and, for $C = \rho^{-1}(\eta)$, $(\texttt{allow}, C, *, \eta_{sid})$ was input at the E-interface before (i.e., with the correct server's nonce for $sid = (\eta, e)$), then output at the sid-sub-interface of the S-interface $(\kappa_C, \eta_{sid}, aux)$.
- Upon input $(\texttt{inject}, \eta, e, aux, \kappa)$ at the E-interface, if $e < e_\eta$, then output at the $sid = (\eta, e_\eta)$-sub-interface of the S-interface $(\kappa, \eta_{sid}, aux)$.

After either a $(\texttt{deliver}, \eta, e, *)$ or a $(\texttt{inject}, \eta, e, *, *)$, if at the S/sid-sub-interface (with $sid = (\eta, e)$) a message $m \in \{0,1\}^{\leq 18432}$ is input, output m at the E-interface. Also, allow at the E-interface to inject messages $m' \in \{0,1\}^{\leq 18432}$ to be output at the S/sid-sub-interface.

Figure 4: The master secret key resource.

In TLS, the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ is constructed in one of three ways: by relying on RSA encryption, by means of a Diffie-Hellman key-exchange protocol with a static server key, and by means of an Ephemeral Diffie-Hellman key exchange, with an ephemeral server key. Next, we assess the security in constructing the master secret key resource using TLS-DH, TLS-DHE, and TLS-RSA.

3.3.1 Diffie-Hellman with a Static Server Key

The DH-based protocol consists of two types of converters, one converter dhc attached to each client interface and one converter $\mathsf{dhs}_{\mathcal{G}}$ attached to the server interface. The protocol is run in a DH-group $\mathbb{G}$ of prime order p, generated by some element g which is chosen when the server certificate is generated, according to some distribution which we denote $\mathcal{G}$. The client converter, dhc, behaves as follows:

1. Obtain the pair (η_C, η_{sid}) of nonces from $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
2. Obtain value *cert* via $\mathsf{SNET}_{\mathsf{N},\rho,n}$, parse *cert* as $((\mathbb{G}, pk_S), s)$.[2] Here we slightly abuse syntax and write $\mathbb{G}$ to mean a description of the group $\mathbb{G}$, including the generator g.[3]
3. Verify the server's certificate by querying $(\texttt{verify}, cert)$ at $\mathsf{PKI}_{\mathfrak{F}}$ (if that fails, abort).
4. Choose a secret $u \in \{1, \dots, |\mathbb{G}|\}$ and send $epk_C = g^u$ via $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
5. Query $pk_S^u|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, receive κ.
6. Output $(\kappa, \eta_C, \eta_{sid}, (epk_C, cert))$.

The server converter, $\mathsf{dhs}_{\mathcal{G}}$, behaves as follows:

0. Sample a group $\mathbb{G} \leftarrow\!\!\$\, \mathcal{G}$ and generate a DH key pair $(pk_S, sk_S) = (g^v, v)$. Register the public key pk_S at $\mathsf{PKI}_{\mathfrak{F}}$ via $(\texttt{register}, (\mathbb{G}, pk_S))$, obtaining the value $cert = ((\mathbb{G}, pk_S), s)$.

For each of the sessions described by a pair (η_C, e):

1. Upon receiving the nonce η_{sid} at the inside interface, respond with the certificate *cert*.
2. Upon receiving $\tilde{epk}_C \in \mathbb{G}$, query $\tilde{epk}_C^v|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, receiving $\tilde{\kappa}$.
3. Output $(\tilde{\kappa}, \eta_{sid}, (\tilde{epk}_C, cert))$ at the outer (η_C, e)-sub-interface.

We prove the following result. The distribution AUX associated with the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ produces two values. The first varies for each session and is an element $\tilde{g} \in \mathbb{G}$ (namely the public key epk_C, where $\mathbb{G}$ is sampled according to $\mathcal{G}$). The second one is fixed for all sessions and is distributed like the certificate *cert*.

Lemma 8. *If the GapDH assumption holds with respect to the distribution $\mathcal{G}$, then the protocol $(\mathsf{dhc}, \mathsf{dhs}_{\mathcal{G}})$ constructs from $[\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}]$ the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$. More formally,*

$$\prod_{C \in \mathcal{C}} \mathsf{dhc}^C \mathsf{dhs}_{\mathcal{G}}{}^S \perp^E [\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}] \quad \equiv \quad \perp^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}, \tag{1}$$

and there is a simulator σ and a reduction $\mathbf{C}$ such that for all distinguishers $\mathbf{D}$, it holds that:

$$\Delta^{\mathbf{D}} \left(\prod_{C \in \mathcal{C}} \mathsf{dhc}^C \mathsf{dhs}_{\mathcal{G}}{}^S [\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}], \sigma^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n} \right) \leq \Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) \,. \tag{2}$$

It might appear surprising that we obtain a tight reduction in equation (2), given that the security statement applies to a setting with multiple parallel sessions. The reason is that we can exploit the random self-reducibility of the GapDH problem to "inject" the $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$-challenge into every session without changing the overall distribution.

Proof.

[2] In the specifications of TLS 1.2, if the public key is not extractable from certificates, for client and/or server, a special message can be sent, which includes the public key. Here we restrict to the standard case.

[3] In particular, the X.509 certificate explicitly encodes the prime order p of the Galois field, the prime sub-order q of the group, and the group generator g, of order q [HFPS99].

Availability. For availability, the cheating bit in the ideal $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ resource is set as $b = 0$, thus the resource outputs the random values as specified and provides channels between the clients' and the server's interfaces that deliver all messages faithfully.

In the real resource, the nonces are exchanged in $\mathsf{SNET}_{\mathsf{N},\rho,n}$, where again the cheating bits are set to $b = 0$. The certificate is correctly generated for the server's public key pk_S, and the client's ephemeral public key epk_C is transmitted correctly. For the output, first note that the key κ, which is output by RO_{384} in the real resource and is picked uniformly at random in the ideal resource, have identical distributions—this is guaranteed by the RO_{384} resource. At the server's sub-interfaces, the consistent keys is delivered (guaranteed by the bit b being set to 0).

Finally, the auxiliary output aux is defined as $(epk_C, cert)$ and has the correct distribution in both cases. Furthermore, once the initial exchanges are done, both resources behave like insecure bidirectional channels in both directions. This verifies condition (1).

Security. The simulator σ does the following:

- Initially, set $e_\eta = 1$ for all $\eta \in \mathcal{N}$ and define an (initially empty) map $R : \{0,1\}^* \to \{0,1\}^{384}$. Choose a function $f \leftarrow \mathfrak{F}$ and a group $\mathbb{G} \leftarrow\!\!\$\, \mathcal{G}$, generate the server's DH key pair $(sk = v, pk = g^v)$ for group $\mathbb{G}$, compute $s = f(\mathbb{G}, pk)$ and output $cert = ((\mathbb{G}, pk), s)$ at the outside interface to simulate the effect of an input $(\mathtt{register}, pk)$ to $\mathsf{PKI}_{\mathfrak{F}}$.

- *Delivery of the client nonce.* Upon input of the type $(\mathtt{ack}, \eta_C)$ at the outside interface, issue $(\mathtt{ack}, \eta_C)$ at the inside interface[4] and receive the generated η_{sid} for $sid = (\eta_C, e_{\eta_C})$. Output η_{sid} at the outside interface as coming from SNET, together with the certificate $cert$, and increase e_{η_C}.

- *Delivery of the server nonce.* Upon input $(\mathtt{deliver}, C, \tilde{\eta})$ at the outside interface and after $cert$ is delivered to C via SNET,[5] generate $u, epk_C = g^u$, and input $(\mathtt{allow}, C, aux, \tilde{\eta})$ at the inside interface (where $aux = (cert, epk_C)$). Simulate sending the DH element epk_C from C via $\mathsf{SNET}_{\mathsf{N},\rho,n}$, and register $(\tilde{\eta}, epk_C)$.

- *Delivery of DH element.* If some group element $\tilde{epk}_C \in \mathbb{G}$ is input at the outside sub-interface $sid = (\eta, e)$ with $e \geq e_\eta$, then:
  - If $\tilde{epk}_C$ corresponds to a recorded pair $(\eta_{sid}, \tilde{epk}_C)$ then issue $(\mathtt{deliver}, \eta, e, aux)$ at the inside interface, with $aux = (cert, epk_C)$;
  - Otherwise, input $(\mathtt{inject}, \eta, e, aux, \kappa)$ at the inside interface, with $aux = (cert, \tilde{epk}_C)$, and with the value κ computed as follows: if $\tilde{epk}_C^v|\mathtt{master\ secret}|\eta|\eta_{sid}$ is defined in R, then use $R(\tilde{epk}_C^v|\mathtt{master\ secret}|\eta|\eta_{sid})$; otherwise define it as a freshly sampled value $R(\tilde{epk}_C^v|\mathtt{master\ secret}|\eta|\eta_{sid}) \leftarrow\!\!\$\, \{0,1\}^{384}$ and use that.

- *Simulation of random oracle.* Whenever the adversary E or either of the honest parties are supposed to query the RO_{384} resource, return a random element $R(x) \leftarrow\!\!\$\, \{0,1\}^{384}$, unless the same input was queried before (in that case, return the value that was previously returned).

[4]For the analysis, note that in the ideal resource the keys output at the client, server, and E interfaces are all independent of the input nonces; thus, it is not important whether the adversary has forwarded an honestly generated η_C or injected a different $\tilde{\eta}_C$.

[5]Note that we simplify the certificate verification step in our analysis, confining ourselves to simply outputting a bit, indicating validity/non-validity. In TLS, this is done in a more thorough manner, and several alert messages are generated, depending on where the verification failed.

Denote $\mathbf{R} \coloneqq \prod_{C \in \mathcal{C}} \mathsf{dhc}^C \mathsf{dhs}_{\mathcal{G}}{}^S[\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}]$ and let $\mathbf{S} \coloneqq \sigma^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$. We first analyze the behavior of $\mathbf{R}$ and $\mathbf{S}$ in the initial queries.

- The exchange of the server's nonces in the real and the ideal resource (the former using SNET for it) is identical, the output at the E-interface having exactly the same distributions.
- The value *cert* has the same distribution in both cases.
- Injecting a value $\tilde{epk}_C$ in a session between client C and the interface $sid = (\eta, e)$, such that the client C has already sent $epk_C \neq \tilde{epk}_C$ and computed its output, does not cause discrepancies. In particular, as the keys are injected by the simulator and the random oracle is simulated consistently, the simulation in this case is perfect, irrespective of any queries the adversary makes to the random oracle.

Intuitively, the discrepancy between the two resources appears when the distinguisher expects a different output from the random oracle resource RO_{384} (in the real world) than the output generated by the simulator (in the ideal world). This occurs (only) in sessions where the key κ output by the client or server is obtained from RO_{384} via a query that is also asked at RO_{384} by the adversary, whereas in the ideal world, the output at the client's or server's interface is picked uniformly at random by $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ and the simulator cannot simulate queries to RO_{384} consistently.

In the following, we show a reduction such that if such an event happens for *some* session, this reduction outputs a solution to the $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$ game (see Appendix C.2)—in particular, the distinguisher must be able to find the input value $\tilde{epk}_C^{\,v}$ by only knowing $epk_C = g^u$ and g^v. To simulate the environment for the distinguisher, the reduction will make "DDH-queries" at $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$, which enables it to simulate the random oracle consistently.

We describe a reduction $\mathbf{C}$ that obtains $(\mathbb{G}, g, g_1, g_2)$ at the inside interface and starts simulating the setup using the group description $(\mathbb{G}, g)$ and g_1 as the server's public key (thus for the simulated instance it holds that $g^v = g_1$), and otherwise behaves similar to $\mathbf{S}$. To simulate a group element sent by the client C_i, $\mathbf{C}$ chooses a value $r_i \in [|\mathbb{G}|]$ uniformly at random and simulates sending $\tilde{g}_i \coloneqq g_2 \cdot g^{r_i}$.

As the reduction does not know the server's secret, i.e. v, it has to make DDH-queries at $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$ to answer random oracle queries consistently. In more detail, the simulation of queries goes as follows. For administrative purposes, the reduction keeps track of what the distinguisher knows about the random oracle by updating two lists, initially empty. One list, which we denote as $\mathcal{H}_H$ records queries of the form (x, η, η_{sid}, y) (the value y is the output to honest parties to input of the form $x|\texttt{master secret}|\eta|\eta_{sid})$. The second list is denoted $\mathcal{H}_M$ and records malicious deliveries of $\tilde{epk}_C$ values by the distinguisher to the simulated server; the entries are of the form $(\tilde{epk}_C, \eta, \eta_{sid}, y)$, where y is the output returned by the reductions when simulating RO_{384} for the maliciously injected input $\tilde{epk}_C$. The queries are answered by the reduction as follows:

- Whenever the random oracle is queried on input (x, η, η_{sid}), first check if there exists an entry $(x, \eta, \eta_{sid}, y) \in \mathcal{H}_H$ and if so, output y; else check if $\mathbf{DDH}(g_1, g_2, x \cdot g_1^{-r_i}) = 1$ for some $i \in \{1, \ldots, |\mathcal{C}|\}$ and if so, output $x \cdot g_1^{-r_i}$ as solution to $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$. Else check for all entries $(\tilde{epk}_C, \eta, \eta_{sid}, y) \in \mathcal{H}_M$ and return y if $\mathbf{DDH}(\tilde{epk}_C, g_1, x) = 1$. Otherwise, respond with a random y and update the list $\mathcal{H}_H \leftarrow \mathcal{H}_H \cup \{(x, \eta, \eta_{sid}, y)\}$.
- When the adversary delivers some group element $\tilde{epk}_C$ to the server session sid, the reduction needs to simulate a random oracle query on input $(\tilde{epk}_C^{\,v}, \eta, \eta_{sid})$, where $v = \log_g g_1$. As the reduction does not know v, it first checks if there is an entry $(\tilde{epk}_C, \eta, \eta_{sid}, y) \in \mathcal{H}_M$; if such an entry exists, the reduction returns y; else, it checks if there exists an input of

the form $(x, \eta, \eta_{sid}, y) \in \mathcal{H}_H$ such that $\mathbf{DDH}(\tilde{epk}_C, g_1, x) = 1$ and if such a value exists, it returns it and updates $\mathcal{H}_M \leftarrow \mathcal{H}_M \cup \{(\tilde{epk}_C, \eta, \eta_{sid}, y)\}$. If no such value exists then the reduction returns a random value y, and also updates $\mathcal{H}_M \leftarrow \mathcal{H}_M \cup \{(\tilde{epk}_C, \eta, \eta_{sid}, y)\}$.

We define MBOs $\mathcal{E}^i = (E_1^i, E_2^i, \dots)$ on the systems $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{C}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}$ as follows: E_q becomes 1 if after q queries there has been a query $(pmk|\mathtt{master\ secret}|\eta_C|\eta_{sid})$ at the random oracle, such that $C = C_i$, $sid = (\eta_C, j)$ for some $j \in [n]$, and pmk is the DH element corresponding g_1 and $\tilde{g}_i$. We then set $\mathcal{E} = \bigvee_{i=1}^{n} \mathcal{E}^i$. Note that for $g_1 = g^u$, $g_2 = g^v$ (so $\tilde{g}_i = g^{v+r_i}$) and $pmk = g^{u(v+r_1)} = g^{uv} \cdot g^{ur_i} = g^{uv} \cdot g_1^{r_i}$, the reduction $\mathbf{C}$ always obtains the correct $g^{uv} = pmk \cdot g_1^{-r_1}$.

Now it holds that $\mathbf{R}^{\mathcal{E}} \overset{g}{\equiv} \mathbf{S}^{\mathcal{E}} \overset{g}{\equiv} \mathbf{C}^{\mathcal{E}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}$, in particular all clients' group elements are uniformly distributed by their definition in dhc, σ, and $\mathbf{C}$. Using Lemma 19 together with the fact that $\mathbf{C}$ wins the game $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$ whenever the output $\mathcal{E}$ is provoked, formally $\Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) = \Gamma^{\mathbf{D}}(\mathbf{C}\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) \geq \Gamma^{\mathbf{D}}\left(\mathbf{C}^{\mathcal{E}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}\right)$, concludes the proof. □

3.3.2 Diffie-Hellman with an Ephemeral Server Key

The construction of the master secret key resource using the ephemeral Diffie-Hellman mode can be proven in two constructive steps. First, we use the signature scheme to construct a network $\rhd\!\!-\!\bullet$ from SNET and PKI; this network allows the server to send one message in each session in an authenticated way. Second, we complete the analysis of the TLS-DHE mode by using the resource $\rhd\!\!-\!\bullet$ to transmit the group parameters and the server's ephemeral public key.

The authenticated transmission resource. The resource $\rhd\!\!-\!\bullet$ has (client) interfaces $C \in \mathcal{C} \subseteq \mathcal{A}_{\mathrm{TCP}}$, a (server) interface S with sub-interfaces $(\eta, e) \in \mathcal{N} \times \mathbb{N}$ with $e \in \mathbb{N}$, and an (eavesdropper) interface E with sub-interfaces $\mathcal{A}_{\mathrm{TCP}} \cup (\mathcal{N} \times \mathbb{N})$, and is parametrized by an injection $\rho : \mathcal{C} \to \mathcal{N}$, a distribution $\mathfrak{F}$ over functions $f : \{0,1\}^* \to \{0,1\}^*$, and a signature scheme $SIG = (gen, sign, vrf)$ as defined in Appendix B.1. The resource is described in detail in Figure 5.

The following protocol constructs $\rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$ from $\mathsf{SNET}_{\mathsf{N},\rho,n}$ and $\mathsf{PKI}_{\mathfrak{F}}$: The client's converter vrf is based on the signature scheme SIG and behaves as follows:

1. Obtain the pair $(\eta_C, \eta_{sid}) \in \mathcal{N}^2$ from $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
2. Obtain the first message $cert$ from $\mathsf{SNET}_{\mathsf{N},\rho,n}$. Query $(\mathtt{verify}, cert)$ at $\mathsf{PKI}_{\mathfrak{F}}$; abort if the verification fails or if $cert$ is not a well-formed certificate $cert = (vk, f(vk))$.
3. Obtain the second message m' from $\mathsf{SNET}_{\mathsf{N},\rho,n}$ and parse m' as (m, s) (abort if that is impossible). If $vrf(\eta_C|\eta_{sid}|m, s; vk) = 1$, then output $(cert, \eta_C, \eta_{sid}, m, s)$ at the outside interface. (Otherwise abort.)
4. Forward all further messages between the inside and the outside interface.

The server's converter sgn provides at the outside "sessions" for all $sid = (\eta_C, e) \in \mathcal{N} \times [n]$ and behaves as follows:

0. Compute $(sk, vk) \leftarrow gen$. Input vk at $\mathsf{PKI}_{\mathfrak{F}}$, obtaining a response s, and set $cert = (vk, s)$. Output $cert$ at the outside interface (as auxiliary information).

For each session $sid = (\eta_C, e)$—i.e., the inputs/outputs at $\mathsf{SNET}_{\mathsf{N},\rho,n}$ are at the corresponding inside sub-interface sid, and the inputs/outputs at the outside interface are at sub-interface sid—do:

The Authenticated Transmission $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$

Initially, generate a key pair $(sk, vk) \leftarrow gen$. Accept at the E-interface an input $b \in \{0,1\}$. If $b = 0$, then:

- Initially, choose $f \leftarrow \mathfrak{F}$. Output $aux = (vk, f(vk))$ at interface S. For each $\eta \in \rho(\mathcal{C})$ and $sid = (\eta, 1)$, sample $\eta_{sid} \leftarrow\!\!\$\ \mathsf{N}$ and output η_{sid} at interface S/sid.
- Upon input the first message m at interface S/sid with $sid = (\eta, 1)$ with $\eta \in \rho(\mathcal{C})$, generate a signature $s \leftarrow sign(\eta|\eta_{sid}|m; sk)$, output s at interface S/sid, and output $(aux, \eta, \eta_{sid}, m, s)$ at interface $C = \rho^{-1}(\eta)$. (For inputs at interfaces $S/(\eta, e)$ with $e > 1$ or $\eta \notin \rho(\mathcal{C})$, no output is generated.)
- All further messages m' input at interface S/sid with $sid = (\rho(C), 1)$ for some $C \in \mathcal{C}$ are output at interface C, and all messages m'' input at C are output at interface S/sid with $sid = (\rho(C), 1)$.

Otherwise, initialize $e_\eta = 1$ for all $\eta \in \mathcal{N}$ and a buffer $\mathcal{B} = \emptyset$. Output vk at the E-interface.

- Initially, obtain $aux \in \{0,1\}^*$ at the E-interface. (This is supposed to contain the certificate.) Output aux at the S-interface.
- Upon input $(\mathtt{ack}, \eta)$ for $\eta \in \mathcal{N}$ at the E-interface, if $e_\eta \leq n$ then draw η_{sid} uniformly at random from $\mathcal{N} \setminus \{\eta_{sid_1}, \ldots, \eta_{sid_{e_{\tilde{\eta}}-1}}\}$ with $sid_i = (\eta, i)$, increase e_η, and output η_{sid} at interfaces S/sid and E/sid.
- Upon input the first message m at interface S/sid (with $sid = (\eta, e)$ and $e < e_\eta$, otherwise ignore the input), generate a signature $s \leftarrow sign(\eta|\eta_{sid}|m; sk)$, record $(\eta|\eta_{sid}|m, s)$ in $\mathcal{B}$, output s at interface S/sid, and (m, s) at interface E/sid. Afterward, if at interface S/sid a message $m \in \{0,1\}^{\leq 18432}$ is input, output m at interface E/sid. Also, allow at interface E/sid to inject messages $m' \in \{0,1\}^{\leq 18432}$ to be output at interface S/sid.
- Upon input $(\mathtt{deliver}, C, \tilde{\eta}, \tilde{m})$ at the E-interface (once for each $C \in \mathcal{C}$), if there is a record $(\eta|\tilde{\eta}|\tilde{m}, \tilde{s}) \in \mathcal{B}$ such that $\eta = \rho(C)$, output $(aux, \eta, \tilde{\eta}, \tilde{m}, \tilde{s})$ at interface C. Afterward, if at the C-interface a message $m \in \{0,1\}^{\leq 18432}$ is input, output m at the interface E/C. Also, allow at interface E/C to inject messages $m' \in \{0,1\}^{\leq 18432}$ to be output at the C-interface.

Figure 5: The network allowing the server to transmit, to each client, one message authentically.

1. Receiving a nonce η_{sid} from $\mathsf{SNET}_{\mathsf{N},\rho,n}$, output η_{sid} at the outside and send *cert* via $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
2. Obtaining a message m at the outside, compute $s \leftarrow sign(\eta_C|\eta_{sid}|m; sk)$ and send (m, s) via $\mathsf{SNET}_{\mathsf{N},\rho,n}$. Output s at the outside.
3. Forward messages between inside and outside.

Lemma 9. *The protocol* $(\mathsf{vrf}, \mathsf{sgn})$ *for a particular signature scheme* $SIG = (gen, sign, vrf)$ *constructs* $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$ *from* $\mathsf{SNET}_{\mathsf{N},\rho,n}$ *and* $\mathsf{PKI}_{\mathfrak{F}}$*, if the signature scheme SIG is unforgeable. More formally,*

$$\prod_{C \in \mathcal{C}} \mathsf{vrf}^C \mathsf{sgn}^S \bot^E [\mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}] \quad \equiv \quad \bot^E \rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n} \tag{3}$$

and there is a simulator σ and a reduction $\mathbf{C}$ such that for all distinguishers $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{vrf}^{C}\mathsf{sgn}^{S}[\mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}], \sigma^{E}\succ\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}\right) \le \Gamma^{\mathbf{DC}}(\mathbf{G}^{\textsf{uf-cma}}). \tag{4}$$

Proof. We first verify condition (3). The keys (sk, vk) and the function f have the same distribution in the real and the ideal case,[6] the same holds for the auxiliary information $aux = (vk, f(vk))$ output at interface S. Each interface S/sid with $sid = (\eta_C, 1)$ and $\rho(C) = \eta_C$ outputs a random nonce η_{sid} also in both cases. On input the first message m at interface S/sid, the same interface outputs a (valid) signature s for m, and the client's interface C outputs $(aux, \eta_C, \eta_{sid}, m, s)$ (since all messages can be parsed correctly, the verification of the server's certificate at $\mathsf{PKI}_{\mathfrak{F}}$ succeeds, and the verification of the signature s succeeds by the correctness of SIG). Afterward, messages input at interface C are output at interface S/sid and vice versa—in the real case these are simply transmitted via $\mathsf{SNET}_{\mathsf{N},\rho,n}$.

For proving condition (4), we consider the following simulator σ:

- Initially, σ obtains the public key vk at the inside interface, chooses a function $f \leftarrow \mathfrak{F}$, and computes $cert = (vk, f(vk))$. Also, σ internally initializes $e_\eta = 1$ for all $\eta \in \mathcal{N}$, and inputs $cert$ as auxiliary information at the inside interface, and as output of $\mathsf{PKI}_{\mathfrak{F}}$ at the outside interface.
- Upon input $(\texttt{ack}, \eta)$ for a nonce $\eta \in \mathcal{N}$ at the outside interface, set $sid = (\eta, e_\eta)$. If $e_\eta \le n$, then input $(\texttt{ack}, \eta)$ at the inside interface. Obtain the nonce η_{sid} at the inside interface, output η_{sid} as response of $\mathsf{SNET}_{\mathsf{N},\rho,n}$ at the outside interface, and increase e_η. Also, output $cert$ at the outside interface as being transmitted via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ in session sid.
- Upon input $(\texttt{deliver}, C, \tilde{\eta})$ at the outside sub-interface corresponding to $\mathsf{SNET}_{\mathsf{N},\rho,n}$, if this is the first such query for C, record the nonce as $\eta_{sid} \coloneqq \tilde{\eta}$ for $sid = (\rho(\eta), *)$.
- When obtaining an output (m, s) at the inside sub-interface sid with $sid = (\eta, e)$, output (m, s) as being transmitted on $\mathsf{SNET}_{\mathsf{N},\rho,n}$ as the second message via the corresponding sub-interface. Mark sid as "active".
- When receiving at the outside sub-interface corresponding to $\mathsf{SNET}_{\mathsf{N},\rho,n}$ the first message $\tilde{m}_1$ to a client C, record C as "failed" if $\tilde{m}_1 \ne cert$.
- When receiving at the outside sub-interface corresponding to $\mathsf{SNET}_{\mathsf{N},\rho,n}$ the second message $\tilde{m}_2 = (m', s')$ to a client C, if C is not marked as "failed" *and*, with $sid = (\eta_C, *)$, $vrf(\eta_C|\eta_{sid}|m', s'; vk) = 1$, then input $(\texttt{deliver}, C, \eta_{sid}, m')$ at the inside interface[7] and mark C as "active".
- Communication is forwarded: for clients $C \in \mathcal{C}$ marked as "active," messages $m \in \mathcal{M}$ obtained at sub-interface C at the inside are output at the outside as being sent by C via $\mathsf{SNET}_{\mathsf{N},\rho,n}$, and messages $m' \in \mathcal{M}$ obtained at the outside as input to $\mathsf{SNET}_{\mathsf{N},\rho,n}$ directed to C are input at sub-interface C at the inside. The analogous behavior applies to the (server-bound) "active" sub-interfaces described by $sid = (\eta, e)$.

Moreover, we use the reduction $\mathbf{C}$ that connects with the inside interface to the game $\mathbf{G}^{\textsf{uf-cma}}$ and provides at the outside interface an emulation of $\mathbf{R} \coloneqq \prod_{C\in\mathcal{C}} \mathsf{vrf}^{C}\mathsf{sgn}^{S}[\mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}]$, using the key pair (sk, vk) obtained from $\mathbf{G}^{\textsf{uf-cma}}$. Roughly, $\mathbf{C}$ emulates $\mathbf{R}$, but computes

[6] In the composed scheme, we let $\mathfrak{F}$ be the distribution that one gets by generating a key pair according to the signature scheme used by the assumed public-key infrastructure, and then for a parameter $x \in \{0,1\}^*$ the result of f is the pair (x, s) with $s \leftarrow sign(x; sk)$.

[7] This has an effect only if the corresponding message has been sent by the server before; if the signature is forged, the real and ideal systems behave differently.

the certificate $cert = (vk, f(vk))$ using the verification key vk obtained from $\mathbf{G}^{\textsf{uf-cma}}$, and the signatures $s \leftarrow sign(\eta_C|\eta_{sid}|m; sk)$ in the converter $\textsf{sgn}$ using signing queries to $\mathbf{G}^{\textsf{uf-cma}}$. A "forgery" in the emulated execution can then be used to win the game $\mathbf{G}^{\textsf{uf-cma}}$ (for the exact definition of forgery see the MBO below; the forgery for $\mathbf{G}^{\textsf{uf-cma}}$ is achieved by concatenating the corresponding nonces and the message).

We define the following MBO $\mathcal{E}$ on the systems $\mathbf{R}$, $\mathbf{S} \coloneqq \sigma^E \rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$, and $\mathbf{CG}^{\textsf{uf-cma}}$ as the following "no forgery"-event: it becomes 1 once there is an input $(\tilde{m}, \tilde{s})$ at the E/sid-interface with $sid = (\eta_C, e)$ such that $vrf(\eta_C|\eta_{sid}|\tilde{m}, \tilde{s}; vk) = 1$—such that η_{sid} was generated as a response in the S-session sid—unless there was an output $(\tilde{m}, \tilde{s}')$ at some interface E/sid' with $sid' = (\eta, e')$ and $\eta_{sid'} = \eta_{sid}$ before. Then, proving "game equivalence" and using Lemma 19 allows to conclude (4).

The equivalence can be seen as follows (we use the counter variables e_η in the same way they are defined in the systems, i.e., counting the number of sessions that have been initiated with nonce η):

- Initially, all systems output the certificate $cert = (vk, f(vk))$, with vk and f chosen according to the same distributions, as auxiliary information at the S-interface, and as an output corresponding to $\mathsf{PKI}_{\mathfrak{F}}$ at the E-interface.
- Upon input $(\texttt{ack}, \eta)$ at the outside E-interface corresponding to $\mathsf{SNET}_{\mathsf{N},\rho,n}$, the system $\mathbf{R}$ outputs a nonce η_{sid} at the E-interface of $\mathsf{SNET}_{\mathsf{N},\rho,n}$ and the S/sid for $sid = (\eta, e_\eta)$, as well as $cert$ being sent via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ from the server-session sid to the client C (see the description of $\textsf{sgn}$). In $\mathbf{S}$, the nonce is generated according to the same distribution and output at the same interfaces; the message $cert$ is simulated correctly.
- Upon input $(\texttt{deliver}, C, \tilde{\eta})$ at the outside E-interface corresponding to $\mathsf{SNET}_{\mathsf{N},\rho,n}$, there is no immediate output (neither for $\mathbf{R}$ nor for $\mathbf{S}$).
- Upon delivering the first message $\tilde{m}_1$ via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ to a client C, if $\tilde{m}_1 \neq cert$ then, in $\mathbf{R}$, $\textsf{vrf}$ aborts since either $\tilde{m}_1$ is not a well-formed certificate or the verification at $\mathsf{PKI}_{\mathfrak{F}}$ fails. In $\mathbf{S}$, σ marks C as "failed" in the same cases. There is no output, neither in $\mathbf{R}$ nor in $\mathbf{S}$.
- Upon the first input m at the outside S/sid-interface with $sid = (\eta, e)$, system $\mathbf{R}$ generates a signature s for $\eta_C|\eta_{sid}|m$ using the key sk (within $\textsf{sgn}$), outputs s at the same sub-interface and the pair (m, s) at the E-interface of $\mathsf{SNET}_{\mathsf{N},\rho,n}$. Within $\mathbf{S}$, the signature is computed analogously by $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$, also output at S/sid, and the message/signature pair is output at the E-interface via σ.
- Upon delivering the second message $\tilde{m}_2$ via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ to a client C, if $\tilde{m}_2$ can be parsed as a pair (m, s) such that, with $sid = (\eta_C, *)$, $vrf(\eta_C|\eta_{sid}|m, s; vk) = 1$, then in $\mathbf{R}$ $(cert, \eta_C, \eta_{sid}, m, s)$ is output at the C-interface (and nothing if $\textsf{vrf}$ aborted earlier or because the verification failed). By the definition of $\mathcal{E}$, this means that there was an output (m, s') at some interface E/sid' with $sid' = (\eta_C, e)$, $\eta_C = \rho(C)$, and $\eta_{sid'} = \eta_{sid}$ (where η_{sid} was input via $(\texttt{deliver}, C, \eta_{sid})$).
  In $\mathbf{S}$, the simulator σ also checks whether the message $\tilde{m}_2$ can be parsed correctly, the client has not "failed," and the signature verifies for the message extended by prepending the nonces η_C and η_{sid} with $sid = (\eta_C, *)$, where the latter nonce was delivered to C as the server's nonce before. The MBO assures that there exists a corresponding record in the buffer $\mathcal{B}$, which means that the output of $\mathbf{R}$ and $\mathbf{S}$ is consistent if no forgery occurs, i.e., the systems are equivalent as games.
- After a certain session (either $C \in \mathcal{C}$ or sid at S) has been initialized, messages input there are output at the corresponding sub-interface of the E-interface and vice versa. This is consistent in both $\mathbf{R}$ and $\mathbf{S}$.

As the same arguments (with the exception that the signatures are obtained from $\mathbf{G}^{\textsf{uf-cma}}$ but have the same distribution) hold for the distribution in the case $\mathbf{CG}^{\mathsf{uf\text{-}cma}}$, and each violation of $\mathcal{E}$ can be used to win $\mathbf{G}^{\mathsf{uf\text{-}cma}}$, this concludes the proof. $\square$

Constructing the key. The subsequent construction step is then achieved by the protocol $(\mathsf{dhec}, \mathsf{dhes}_{\mathcal{G}})$, in which the server chooses for each session a (potentially fresh) Diffie-Hellman group and element, which are sent as an authenticated message via $\rhd\!\!-\!\bullet$. We denote the distribution over groups that the server uses by $\mathcal{G}$; the only restriction implied by the TLS standard is that the group specified as $\mathbf{Z}_p^\times$, where p is represented by at most 65535 bits [DR08, page 51].[8] We write $\mathsf{dhes}_{\mathcal{G}}$ wherever we want to make the distribution $\mathcal{G}$ used by the server converter explicit.

The distribution AUX in this case consists of two parts. The first part is the same for all sessions and consists of the certificate (i.e., depends on $\mathfrak{F}$ and SIG of $\rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$). The distribution is described by $(sk, vk) \leftarrow gen$, $f \leftarrow\$\, \mathfrak{F}$, and then $cert = (vk, f(vk))$. The second part is chosen independently for each session by doing the same process as in the protocol: choose a prime $p \in \mathbb{N}$ and a generator $g \in \mathbf{Z}_p^\times$ according to the distribution $\mathcal{G}$, and choose two group elements $g_1, g_2 \leftarrow\$\, \mathbf{Z}_p^\times$ uniformly at random.
The client's converter dhec obtains $(aux, \eta_C, \eta_{sid}, m, s)$ at the inside interface, and then:

- Parse the message as $p|g|g' = m$ (abort it impossible).
- Choose $u \leftarrow\$\, \{1, \ldots, q\}$ (with $q = |\mathbf{Z}_p^\times|$) and input g^u at the inside interface.
- Query ${g'}^u|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, in order to obtain a key $\kappa \in \{0,1\}^{384}$. Output $(\kappa, \eta_C, \eta_{sid}, aux|m|s|g^u)$.
- Forward the following communication between the inside and the outside interfaces.

The server's converter $\mathsf{dhes}_{\mathcal{G}}$ connects to the S-interface of $\rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$. Both the inside and outside interfaces have sub-interfaces $sid = (\eta_C, e) \in \mathcal{N} \times [n]$. The converter behaves as follows:

- Initially, receive aux on the inside interface.
- Upon obtaining a nonce η_{sid} at sub-interface $sid = (\eta_C, e)$, choose a modulus $p \in \mathbb{N}$ and a generator $g \in \mathbf{Z}_p^\times$ according to the distribution $\mathcal{G}$. Also, choose an exponent $v \leftarrow\$\, \{1, \ldots, |\mathbf{Z}_p^\times|\}$. Send $m = p|g|g^v$ via the inside sub-interface sid. Obtain the signature s at the inside interface in return.
- Upon receiving a group element $\tilde{g}$ in an (active) session $sid = (\eta_C, e)$ at the inside interface, query $\tilde{g}^v|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, call the result κ. Output $(\kappa, \eta_{sid}, aux|m|s|\tilde{g})$ at the outside sub-interface sid.
- Forward the following communication between the corresponding sub-interfaces sid of the inside and the outside interface.

The described protocol indeed constructs the master secret key resource from the network $\rhd\!\!-\!\bullet$ and the random oracle RO_{384} under the assumption that the GapDH assumption holds with respect to the distribution $\mathcal{G}$.

Lemma 10. *The protocol* $(\mathsf{dhec}, \mathsf{dhes}_{\mathcal{G}})$ *constructs from* RO_{384} *and* $\rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$ *the resource* $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$*, under the GapDH assumption for* $\mathcal{G}$*. More formally, with the above-described converters* dhec *and* $\mathsf{dhes}_{\mathcal{G}}$*,*

$$\prod_{C \in \mathcal{C}} \mathsf{dhec}^C \mathsf{dhes}_{\mathcal{G}}{}^S \perp^E [\rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}, \mathsf{RO}_{384}] \quad \equiv \quad \perp^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}, \tag{5}$$

[8] We assume that the server picks a safe prime of appropriate size.

and there is a simulator σ and a reduction $\mathbf{C}$ such that for all distinguishers $\mathbf{D}$

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}}\mathsf{dhec}^{C}\mathsf{dhes}_{\mathcal{G}}{}^{S}[{\rhd\!-\!\bullet}_{\mathsf{N},\rho,\mathfrak{F},SIG,n},\mathsf{RO}_{384}],\sigma^{E}\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}\right)\quad\leq\quad n\cdot|\mathcal{C}|\cdot\Gamma^{\mathbf{DC}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right). \tag{6}$$

Proof. We first argue that equation (5) holds. The "ideal" system $\perp^{E}\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ chooses, for each session $sid=(\eta_C,1)$ with $\rho(C)=\eta_C$, a nonce η_{sid}, a key κ_C and auxiliary information aux_C, and outputs $(\kappa_C,\eta_{sid},aux_C)$, and afterward forwards communication between C and S/sid.

The distribution in the case $\prod_{C\in\mathcal{C}}\mathsf{dhec}^{C}\mathsf{dhes}_{\mathcal{G}}{}^{S}\perp^{E}[{\rhd\!-\!\bullet}_{\mathsf{N},\rho,\mathfrak{F},SIG,n},\mathsf{RO}_{384}]$ is as follows. The key κ_C for each client C is a uniformly random 384-bit string (the random oracle RO_{384} is queried at distinct places as η_C is distinct by the assumption on ρ), the nonces η_{sid} for $sid=(\eta_C,1)$ are distributed according to N, and the auxiliary information aux_C has exactly the same distribution as well. Moreover, after at both interfaces C and S/sid the above information is output, the resource behaves as a bidirectional channel between those interfaces. This verifies equation (5).

The simulator basically needs to take care of the ${\rhd\!-\!\bullet}_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$'s E-interface (beginning with choosing some good aux), and then needs to simulate the DH exchange, i.e., the group and elements generated by the server as well as the element generated by the client. The simulator σ behaves as follows:

- Throughout, σ keeps counters e_η for each $\eta\in\mathcal{N}$ in the usual way (i.e., increase e_η whenever the nonce η is delivered to the server). Also, σ keeps a map $R:\{0,1\}^*\to\{0,1\}^{384}$ which is initially empty.
- Initially, obtain a string $cert\in\{0,1\}^*$ at the outside interface (this is needed for the auxiliary information).
- Upon input $(\mathtt{ack},\eta)$ at the outside, input $(\mathtt{ack},\eta)$ at the inside and obtain as response η_{sid} for $sid=(\eta,e_\eta)$. Output η_{sid} at the outside interface. Also, choose a modulus $p_{sid}\in\mathbb{N}$ and a generator $g_{sid}\in\mathbf{Z}_{p_{sid}}^{\times}$ according to $\mathcal{G}$, as well as a value $v \leftarrow\!\!\$\ \{1,\dots,q_{sid}\}$ for $q_{sid}=|\mathbf{Z}_{p_{sid}}^{\times}|$, compute $\tilde{g}_{sid}=g_{sid}^{v}$ and $s\leftarrow sign(\eta|\eta_{sid}|p_{sid}|g_{sid}|\tilde{g}_{sid};sk)$, and output $(p_{sid}|g_{sid}|\tilde{g}_{sid},s)$ at the outside sub-interface sid.
- Upon input $(\mathtt{deliver},C,\tilde{\eta},\tilde{m})$ at the outside, if $(\mathtt{ack},\eta_C)$ was input before, and there is a session $sid=(\eta_C,e)$ with $\tilde{\eta}=\eta_{sid}$ and $\tilde{m}=p_{sid}|g_{sid}|\tilde{g}_{sid}$, then input $(\mathtt{allow},C,aux,\eta_{sid})$ with $aux=cert|p_{sid}|g_{sid}|\tilde{g}_{sid}|\bar{g}_{sid}$ where $\bar{g}_{sid}=g^{u}$ for a uniformly random $u\in\{1,\dots,q_{sid}\}$. Simulate $\bar{g}_{sid}$ as the first message transmitted from C to sid. (If any check fails, output nothing—note that we simplify the protocol and do not handle error messages.)
- If a group element g'_{sid} is delivered (i.e., input at the outside interface) to some instance sid (and g'_{sid} is a valid group element in the group with modulus p_{sid}):
  - if $g'_{sid}=\bar{g}_{sid}$, then input $(\mathtt{deliver},\eta_C,e,aux)$ and $aux=cert|p_{sid}|g_{sid}|\tilde{g}_{sid}|\bar{g}_{sid}$ (with $sid=(\eta_C,e)$) at the inside interface;
  - otherwise, for $x={g'_{sid}}^{v}|\mathtt{master\ secret}|\eta_C|\eta_{sid}$, if $R(x)$ is undefined, initialize it as $R(x)\leftarrow\!\!\$\ \{0,1\}^{384}$ and input $(\mathtt{inject},\eta_C,e,aux,R(x))$ at the inside interface with $sid=(\eta_C,e)$ and $aux=cert|p_{sid}|g_{sid}|\tilde{g}_{sid}|g'_{sid}$.
- Simulate the random oracle, that is, on input x intended for RO_{384} at the E-interface, if $R(x)$ is not defined then set $R(x)\leftarrow\!\!\$\ \{0,1\}^{384}$. Return $R(x)$.
- Deliver messages faithfully for the sessions where the setup is complete, i.e., for clients $C\in\mathcal{C}$ after a successful—i.e., one that was simulated before—$(\mathtt{deliver},C,\tilde{\eta},\tilde{m})$ query, and for server sessions $sid=(\eta,e)$ after delivering a valid group element (valid with respect to the group used in that session) in response to the authenticated message via ${\rhd\!-\!\bullet}_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$.

We first observe that the two systems $\mathbf{R} = \prod_{C \in \mathcal{C}} \mathsf{dhec}^C \mathsf{dhes}_{\mathcal{G}}{}^S [\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}, \mathsf{RO}_{384}]$ and $\mathbf{S} = \sigma^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ are equivalent with respect to how they treat nonces; in particular, the responses to $(\mathtt{ack}, \eta)$ are determined by choosing fresh parameters in both cases. The same argument holds for the queries in which messages are forwarded in the sessions that completed the setup.

For the queries $(\mathtt{deliver}, C, \eta, m)$, for delivering group elements to a session *sid* of the server, and for querying the random oracle RO_{384}, the difference between $\mathbf{R}$ and $\mathbf{S}$ is that in $\mathbf{R}$, all "keys" output at either $C \in \mathcal{C}$ or $S/(\eta_C, e)$ for $\eta_C \in \mathcal{N}$ and $e \in [n]$ are chosen consistently with the random oracle queries, whereas in $\mathbf{S}$ they are not (in cases where both group elements have been simulated).

We then describe reduction systems $\mathbf{C}_{i,j}$ for $i \in [|\mathcal{C}|]$ and $j \in [n]$, which obtain at the inside interface a modulus p, a generator g, and two group elements g^a, g^b. We assume that there is some (e.g., lexicographic) ordering on the set $\mathcal{C}$, i.e., the elements are $C_1, \ldots, C_{|\mathcal{C}|}$. All systems $\mathbf{C}_{i,j}$ manage internally a map $R : \{0,1\}^* \to \{0,1\}^{384}$ which is initially empty and is managed as a "random oracle with lazy evaluation", i.e., whenever a look-up $R(x)$ for $x \in \{0,1\}^*$ in R fails, the system will choose a fresh random value $y \in \{0,1\}^{384}$, store $R(x) = y$, and use y as the result. The system $\mathbf{C}_{i,j}$ then behaves as follows (we stress that every query can be associated with a (client's) nonce $\eta \in \mathcal{N}$, either because the nonce is given explicitly, or because the query belongs to a session that is described by a nonce and a counter):

- for queries (at the S- and E-interfaces) that are related to nonces $\eta \in \mathcal{N} \setminus \rho(\mathcal{C})$, the system $\mathbf{C}_{i,j}$ can easily reproduce the behavior of the real or ideal systems (their behavior is equivalent for those queries);
- For queries that are related to a client C_ℓ resp. the nonce $\eta_\ell = \rho(C_\ell)$ with $\ell \neq i$, or alternatively to (server) sessions $(\rho(C_i), e)$ with $e \neq j$, emulate the entire sessions. Choose the key output at C_ℓ and the corresponding sub-interface of the server (if there is a consistent session, corresponding to the use of $\mathtt{deliver}$ in $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$) by computing as in the protocol and then evaluating R.
- For the jth $(\mathtt{ack}, \rho(C_i))$-query, emulate the server's response using the parameters (i.e., the modulus p, the generator g, and the first group element g^a) obtained at the inside interface.
- For the query $(\mathtt{deliver}, C_i, \tilde{\eta}, \tilde{m})$ at the sub-interface corresponding to $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$:
  - if $\tilde{\eta} = \eta_{sid}$ for $sid = (\rho(C_i), j)$, and if the message would be admitted by $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$, then emulate the second group element g^b obtained at the inside interface as a message from C_i to *sid*. Choose $\kappa_{C_i} \in_R \{0,1\}^{384}$ and emulate it at interface C_i.
  - if $\tilde{\eta} = \eta_{sid}$ for $sid = (\rho(C_i), e)$ and $e \neq j$, and if the message would be admitted by $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$, then emulate a uniformly chosen element of the group described in $\tilde{m}$; choose the key κ_{C_i} using the map R evaluated on $\tilde{g}|\mathtt{master\ secret}|\rho(C_i)|\eta_{sid}$, where $\tilde{g}$ is the DH key that can be computed because $\mathbf{C}_{i,j}$ chose all parameters.
- Upon delivery of the first client's message in session $sid = (\rho(C_i), j)$, if the message completes a consistent session with C_i, then emulate the output κ_{C_i} at the *sid*-interface. Otherwise, compute the server's output according to the protocol.
- Queries $x \in \{0,1\}^*$ to RO_{384} are answered by evaluating $y = R(x)$ as described above and answering with y. If $x = \tilde{g}|\mathtt{master\ secret}|\rho(C_i)|\eta_{sid}$, where $\tilde{g}$ is a valid group element with respect to the group $\mathbf{Z}_p$, use the queries at the inside interface to determine whether $g^{ab} = \tilde{g}$ and, in case of success, input $\tilde{g}$ as solution at the inside interface and halt.

In the following, we use $\mathbf{C}_{*,*}$ wherever the indices i, j of $\mathbf{C}_{i,j}$ are irrelevant. Consider now the following MBOs $\mathcal{E}^{i,j} = (E_1^{i,j}, E_2^{i,j}, \ldots)$ for each pair (i,j) with $i \in [|\mathcal{C}|]$ and $j \in [n]$, such that

$E_q^{i,j}$ is defined over the random systems $\mathbf{C}_{*,*}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}$ (and $\mathbf{R}$, $\mathbf{S}$):[9] The MBO becomes 1 if after q queries there has been a query $(pmk|\texttt{master secret}|\eta_C|\eta_{sid})$ at the random oracle, such that $C = C_i$, $sid = (\eta_C, j)$ and pmk is the DH element corresponding to session (i,j), i.e., the DH element with respect to the server's group element sent in session (η_C, j) and the group element sent by C_i (given they are in the same group). Then define the MBO $\mathcal{E}$ via $E_q \coloneqq \bigvee_{i,j} E_q^{i,j}$.

By definition of $\mathbf{C}_{i,j}$ and $\mathcal{E}$, one can verify that $\mathbf{R}^{\mathcal{E}} \overset{g}{\equiv} \mathbf{C}_{i,j}^{\mathcal{E}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-} \overset{g}{\equiv} \mathbf{S}^{\mathcal{E}}$ for all pairs $(i,j) \in [|\mathcal{C}|] \times [n]$. Also, provoking the MBO $\mathcal{E}^{i,j}$ in $\mathbf{C}_{i,j}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}$ implies that the reduction $\mathbf{C}_{i,j}$ is successful in winning $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$, so

$$\Gamma^{\mathbf{D}\mathbf{C}_{i,j}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) = \Gamma^{\mathbf{D}}(\mathbf{C}_{i,j}\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) \geq \Gamma^{\mathbf{D}}\left(\mathbf{C}_{i,j}^{\mathcal{E}^{i,j}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}\right)$$

and

$$\sum_{i,j} \Gamma^{\mathbf{D}}\left(\mathbf{C}_{i,j}^{\mathcal{E}^{i,j}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}\right) \geq \Gamma^{\mathbf{D}}\left(\mathbf{C}_{*,*}^{\mathcal{E}}\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}\right)$$

by Lemma 20. The statement then follows using Lemma 19 and using the reduction $\mathbf{C}$ that chooses any one of the $\mathbf{C}_{i,j}$ with $i \in [|\mathcal{C}|]$ and $j \in [n]$ uniformly at random. □

3.3.3 The RSA-PKCS Scheme

The RSA-PKCS-based protocol consists of the following two converters, which are based on the RSA algorithms $RSA = (gen, enc, dec)$ as specified in PKCS#7 [Kal98] (see also Appendix A.2). The client's converter rsac behaves as follows:

1. Obtain the pair (η_C, η_{sid}) of nonces from $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
2. Obtain a message $cert = (pk, s)$ (supposed to be the server's certificate) via $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
3. Verify the server's certificate by querying $(\texttt{verify}, cert)$ at $\mathsf{PKI}_{\mathfrak{F}}$ (if that fails, abort).
4. Choose a secret $pmk \in \{0,1\}^{384}$ as follows: concatenate the two-byte protocol version identifier with a 46-byte uniformly random string, encrypt with the server's public key pk obtained from $cert$ (resulting in a ciphertext $c = enc(pmk; pk)$), and send c via $\mathsf{SNET}_{\mathsf{N},\rho,n}$.
5. Query $pmk|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, call the result κ.
6. Output $(\kappa, \eta_C, \eta_{sid}, (cert, c))$ at the outside interface.
7. Forward all further communication between the outside interface and $\mathsf{SNET}_{\mathsf{N},\rho,n}$.

The server's converter rsas behaves as follows:

0. Generate an RSA key pair $(pk, sk) = gen$ and register the public key pk at $\mathsf{PKI}_{\mathfrak{F}}$ via $(\texttt{register}, pk)$, obtaining a value s, define $cert = (pk, s)$.

For each of the sessions described by a pair $sid = (\eta_C, e) \in \mathcal{N} \times [n]$:

1. Upon receiving a nonce η_{sid} at sub-interface $sid = (\eta_C, e)$, respond with the certificate $cert$.
2. Upon receiving a ciphertext c, decrypt $\widetilde{pmk} = dec(c; sk)$. If $\widetilde{pmk} = \perp$ or if the first two bytes of $\widetilde{pmk}$ do not match the two-byte protocol version identifier, choose a fresh value for $\widetilde{pmk}$ as in Step 4. of rsac.
3. Query $\widetilde{pmk}|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384} and call the result $\tilde{\kappa}$.

[9] The term $\left(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}\right)^{-}$ refers to the game $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$ without its Monotone Binary Output (MBO), see the last paragraph of the proof.

4. Output $(\tilde{\kappa}, \eta_{sid}, (cert, c))$ at the (sid-sub-interface of the) outside interface.
5. Forward all further communication between the (sid-sub-interface of the) outside interface and (the sid-sub-interface of) $\mathsf{SNET}_{\mathsf{N},\rho,n}$.

We show that the described protocol is secure under the NR-PCA assumption for PKCS#7, i.e., it constructs the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$. Here, the distribution AUX in this case consists of two parts. The first part is the same for all sessions and consists of the certificate (i.e., depends on $\mathfrak{F}$ of $\mathsf{PKI}_{\mathfrak{F}}$). The distribution is described by $(sk, pk) \leftarrow gen$, $f \twoheadleftarrow\!\!\$\ \mathfrak{F}$, and then $cert = (pk, f(pk))$. The second part is chosen independently for each session by doing the same process as in the protocol: choose $pmk \in \{0,1\}^{384}$ (as it is done in Step 4. of rsac) and compute $c = enc(pmk; pk)$.

Lemma 11. *The protocol* $(\mathsf{rsac}, \mathsf{rsas})$ *constructs from* $[\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}]$ *the master secret key resource* $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$*, under the NR-PCA assumption for* $RSA = (gen, enc, dec)$*.*

More formally,

$$\prod_{C\in\mathcal{C}} \mathsf{rsac}^C \mathsf{rsas}^S \bot^E [\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}] \quad \equiv \quad \bot^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}, \tag{7}$$

and there is a simulator σ *and reductions* $\mathbf{C}_q$*, for* $q \in \mathbb{N}$*, such that for all distinguishers* $\mathbf{D}$*,*

$$\Delta_q^{\mathbf{D}} \left(\prod_{C\in\mathcal{C}} \mathsf{rsac}^C \mathsf{rsas}^S [\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}], \sigma^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n} \right) \leq n \cdot |\mathcal{C}| \cdot \Gamma_q^{\mathbf{D}\mathbf{C}_q}(\mathbf{G}^{\mathsf{nr\text{-}pca}}) + \frac{q}{2^{368}}. \tag{8}$$

Before proceeding with the formal proof, let us discuss some intuition. In the proofs of Lemma 8 and Lemma 10, it is apparent when the pre-master secret value embedded in the challenge to the reduction is delivered to the server. However, as we argue below, this is not necessarily the case for TLS-RSA if one assumes like [JK02, KPW13] that RSA is OW-PCA. Recall that in the RSA mode, the client chooses a random pre-master secret and encrypts it under the server's certified public key. Now, the argument would be that if the distinguisher can distinguish between the $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ resource and the construction based on the RSA encryption, then it can break the OW-PCA encryption (essentially it knows the pre-master secret). The reduction to OW-PCA uses a "testing" $\mathbf{PCA}(\cdot,\cdot)$ oracle, such that $\mathbf{PCA}(m, c)$ outputs 1 iff. $dec(c; sk) = m$; in our case, this should permit the reduction to properly simulate the random oracle.

In the case of RSA, there is a subtle problem which does not appear in the DH case. In particular, whereas DH elements are not re-randomizable, if the OW-PCA encryption scheme is re-randomizable, the distinguisher between two subsequent hybrids can inject a re-randomized ciphertext which cannot be simulated accurately by the reduction. Recall that in the DHE case, the hybrids were indexed by sessions, such that the input from the GapDH game was inserted into a different session each time. If the distinguisher receives a challenge ciphertext from the OW-PCA reduction, then re-randomizes it and sends it to the server in the "correct session," then the behavior of the two resources will differ, since for one of them the key output of the client and server will coincide, whereas for the other, it will differ (recall that since the randomized ciphertext will decrypt to the same plaintext as the challenge ciphertext, the pre-master secret input into the random oracle is the same). For this reason, similar to [BFK+13b], Lemma 11 above relies on the assumption that RSA-PKCS is not re-randomizable.[10]

[10] An alternative approach would be to be less modular and consider the entire protocol as a single unit, since only the finished messages authenticate the exact ciphertext transmitted during the session. This is essentially the path taken by Krawczyk *et al.* [KPW13].

Proof. We first argue that the availability condition holds. The "ideal" system $\bot^E\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ chooses for each session $sid = (\rho(C), 1)$ a nonce η_{sid}, a key κ_C and auxiliary information aux_C, and outputs $(\kappa_C, \rho(C), \eta_{sid}, aux_C)$ at the C interface and $(\kappa_C, \eta_{sid}, aux_C)$ at the interface S/sid; afterwards the communication is forwarded between C and S/sid.

The distribution in the case $\prod_{C\in\mathcal{C}} \mathsf{rsac}^C \mathsf{rsas}^S[\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}]$ is as follows. The key κ_C for each client C is a uniformly random 384-bit string (the random oracle RO_{384} is queried at distinct places as $\eta_C = \rho(C)$ is distinct by the assumption on ρ), the nonces η_{sid} are chosen according to the distribution N, and the auxiliary information aux_C has exactly the same distribution as well. Moreover, after at both interfaces C and $S/(\eta_C, 1)$ the above information is output, the resource behaves as a bidirectional channel between those interfaces. This verifies equation (7).

In order to prove equation (8), consider the following simulator $\sigma = \sigma(\rho)$ (parametrized by the bijection ρ of $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$):

- Initially, σ sets $e_\eta = 1$ for all $\eta \in \mathcal{N}$ and defines an (initially empty) map $R : \{0,1\}^* \to \{0,1\}^{384}$.
- Choose a function $f \leftarrow \mathfrak{F}$, sample $(pk, sk) \leftarrow gen$, compute $s = f(pk)$ and output $cert = (pk, s)$ at the outside interface to simulate the answer of an input $(\texttt{register}, pk)$ to $\mathsf{PKI}_{\mathfrak{F}}$.
- Upon input $(\texttt{ack}, \eta)$ for a nonce $\eta \in \mathcal{N}$ at the outside interface, set $sid = (\eta, e_\eta)$. If $e_\eta \le n$, then issue $(\texttt{ack}, \eta)$ at the inside interface. Obtain the nonce η_{sid} at the inside interface and output η_{sid} as response of $\mathsf{SNET}_{\mathsf{N},\rho,n}$ at the outside interface; increment e_η. Also, output $cert$ at the outside interface as being transmitted via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ from sid.
- Upon input $(\texttt{deliver}, C, \tilde{\eta})$ and after $cert$ has been delivered via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ at the outside interface, issue $(\texttt{allow}, C, aux, \tilde{\eta})$ at the inside interface. The auxiliary information aux consists of $aux = (cert, c)$ where the ciphertext c is computed by encrypting a value pmk (as in Step 4. of rsac) using public key pk. Output c at the outside interface as being transmitted via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ from C and register $(C, \tilde{\eta}, pmk)$.
- When obtaining a ciphertext $\tilde{c}$ at the outside sub-interface sid, with $sid = (\eta, e)$, compute $\widetilde{pmk} = dec(\tilde{c}; sk)$. If $dec(\tilde{c}; sk) = \bot$ or if the first two bytes of $\widetilde{pmk}$ do not match the two-byte protocol version identifier, then choose $\widetilde{pmk}$ uniformly at random instead. Then:
  - If $\eta \in \rho(\mathcal{C})$ and with $C = \rho^{-1}(\eta)$, in case $\widetilde{pmk}$ corresponds to a previously recorded triple $(C, \eta_{sid}, \widetilde{pmk})$, input $(\texttt{deliver}, \eta, e, aux)$ at the inside interface, with aux being $(cert, \tilde{c})$.
  - Otherwise, input $(\texttt{inject}, \eta, e, aux, \kappa)$ at the inside interface, with $aux = cert|\tilde{c}$ with the ciphertext $\tilde{c}$ just injected. The value κ is defined as $R(\widetilde{pmk}|\texttt{master secret}|\eta|\eta_{sid})$. (If $\widetilde{pmk}|\texttt{master secret}|\eta|\eta_{sid}$ has not been queried to the random oracle before, define it as $R(\widetilde{pmk}|\texttt{master secret}|\eta|\eta_{sid}) \leftarrow\!\!\$\ \{0,1\}^{384}$.)
- Simulate the random oracle, that is, on input x intended for RO_{384} at the E-interface, if $R(x)$ is not defined then set $R(x) \leftarrow\!\!\$\ \{0,1\}^{384}$. Return $R(x)$.

Define $\mathbf{R} \coloneqq \prod_{C\in\mathcal{C}} \mathsf{rsac}^C \mathsf{rsas}^S[\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}]$ and $\mathbf{S} \coloneqq \sigma^E\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$. We start by noting that the behavior of the two systems $\mathbf{R}$ and $\mathbf{S}$ is identical for the initial queries:

- The value $cert$ has the same distribution in both resources.
- The responses to $(\texttt{ack}, \eta_C)$ are determined by choosing fresh parameters in both cases.

- The distribution corresponding to messages forwarded in the sessions that completed the setup is also the same in the two systems.

For the queries $(\texttt{deliver}, C, \tilde{\eta})$, for delivering a ciphertext to a session *sid* of the server, and for querying the random oracle RO_{384}, the difference between $\mathbf{R}$ and $\mathbf{S}$ is that in $\mathbf{R}$ all "keys" output at either $C \in \mathcal{C}$ or $S/(\eta, e)$ for $\eta \in \mathcal{N}$ and $e \in [n]$ are chosen consistently with the random oracle queries, whereas in $\mathbf{S}$ they are not (in cases where the ciphertext has been simulated). The intuition behind the proof is that a distinguisher telling apart the two systems would have to query the random oracle at such a point; however, as we argue below, the latter is not very likely to happen as otherwise such a distinguisher can be used to break the NR-PCA assumption for $RSA = (gen, enc, dec)$ (see Appendix C.1). Recall that in this game the challenger produces a ciphertext c^* corresponding to the encryption of a random message m^* using public key pk, and the goal of the adversary is to produce a re-randomized ciphertext $c' \neq c^*$ decrypting to m^*, with the help of a $\mathbf{PCA}(\cdot, \cdot)$ oracle (where $\mathbf{PCA}(m, c)$ outputs 1 iff $dec(c; sk) = m$).[11]

We now prove equation (8). We describe a series of reductions $\mathbf{C}_{i,j,q}$ with $i \in [|\mathcal{C}|]$, $j \in [n]$, and $q \in \mathbb{N}$, where the indices pinpoint a session run for a nonce η corresponding to a client $C \in \mathcal{C}$ and a corresponding nonce η_{sid} generated for session $sid = (\eta, e)$, where $1 \leq e \leq e_\eta \leq n$. We assume some (e.g. lexicographic) order $\preceq$ over the set of clients, and write C_i to denote the i-th client with respect to this order. The reduction $\mathbf{C}_{i,j,q}$ uses the public key pk and injects the challenge c^* received from $\mathbf{G}^{\textsf{nr-pca}}$, relies on the PCA queries it can make at $\mathbf{G}^{\textsf{nr-pca}}$, and it works as follows:

- Let (pk, c^*) be the input received from the inside interface with $\mathbf{G}^{\textsf{nr-pca}}$.
- For all the sessions between a client $C \neq C_i$ and for the sessions of C_i with server interfaces (η, e) such that $e \neq j$, $\mathbf{C}_{i,j,q}$ uses pk to encrypt a pmk and generates the clients' keys as in the real resource.
- In the session between client C_i and the server session (η, j), the session is emulated using c^*. In this case the output at the client and server interface is independent of the queries made to RO_{384}.

The simulation of the random oracle queries and server master secret keys goes into more details as follows. Each of the reductions will keep two (initially empty) lists $\mathcal{H}_P$ and $\mathcal{H}_C$. List $\mathcal{H}_P$ contains entries of the form $(pmk, \eta_C, \eta_{sid}, y)$ while $\mathcal{H}_C$ contains entries of the form $(c, \eta'_C, \eta'_{sid}, y')$. $\mathcal{H}_P$ corresponds to random oracle queries, while $\mathcal{H}_C$ corresponds to master secret keys output at the server. The two lists need to be kept consistent, in the sense that if for two entries $\eta_C = \eta'_C$, $\eta_{sid} = \eta'_{sid}$, and $\mathbf{PCA}(pmk, c) = 1$ then $y = y'$. Hence:

- Whenever the random oracle is queried on input $pmk|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, first look for an entry $(pmk, \eta_C, \eta_{sid}, y)$ in $\mathcal{H}_P$; return the corresponding value y in case such entry is found. Check for the entry $(\tilde{c}, \eta_C, \eta_{sid}, y)$ in $\mathcal{H}_C$, and return the corresponding y if $\mathbf{PCA}(pmk, \tilde{c}) = 1$.

  Otherwise, check if $\mathbf{PCA}(pmk, c^*) = 1$:

  - If this is the case, run $c' \leftarrow enc(pmk; pk)$ with different randomness until $c' \neq c^*$ and output c' as the re-randomized ciphertext in $\mathbf{G}^{\textsf{nr-pca}}$; note that, according to PKCS#7, the above re-sampling requires to encrypt the plaintext with a different random pad P' and two trials are sufficient.

[11] Note that in TLS the message space is of the form $\mathcal{M} = \texttt{version\_number} \times \{0,1\}^{368}$. The NR-PCA assumption over this space is implied by the one on $\{0,1\}^{386}$, but one would loose an additional term $\frac{1}{2^{16}}$ in the reduction.

  - Else, respond with a random y and update the list $\mathcal{H}_P$ accordingly.

- When some ciphertext $\tilde{c}$ is delivered to a server's session $sid = (\eta, e)$, search for an entry $(\tilde{c}, \eta, \eta_{sid}, y)$ in $\mathcal{H}_C$. Otherwise, check whether there is an entry $(\tilde{pmk}, \eta, \eta_{sid}, y)$ in $\mathcal{H}_P$ such that $\mathbf{PCA}(\tilde{pmk}, \tilde{c}) = 1$. Return the corresponding value y in case either of these entries is found; else, respond with a random y and update the list $\mathcal{H}_C$ accordingly.

- If after q queries no ciphertext c' has been returned to $\mathbf{G}^{\mathsf{nr\text{-}pca}}$, check whether $\mathcal{H}_C$ contains an entry $(c, \eta_C, \eta_{sid}, *)$ such that $C = C_i$, $sid = (\eta_C, j)$ and return c to $\mathbf{G}^{\mathsf{nr\text{-}pca}}$.

As for DHE, we use $\mathbf{C}_{*,*,q}$ wherever the indices i, j of $\mathbf{C}_{i,j,q}$ are irrelevant and we consider a monotone binary output (MBO) $\mathcal{E}^{i,j} = (E_1^{i,j}, E_2^{i,j}, \dots)$ for each pair (i, j) with $i \in [|\mathcal{C}|]$ and $j \in [n]$. Here, $E_q^{i,j}$ is defined over the random systems $\mathbf{C}_{*,*,q'}\,(\mathbf{G}^{\mathsf{nr\text{-}pca}})^-$ (and $\mathbf{R}$, $\mathbf{S}$):[12] The MBO becomes 1 if after q queries there has been a query $(pmk|\texttt{master secret}|\eta_C|\eta_{sid})$ at the random oracle, such that $C = C_i$, $sid = (\eta_C, j)$ and $dec(c^*, sk) = pmk$ or a ciphertext c has been delivered in session $sid = (\eta_C, j)$ with $C = C_i$ such that $dec(c^*, sk) = dec(c; sk)$.[13]

We also define three MBOs $\mathcal{F} = (F_1, F_2, \dots)$, $\mathcal{F}' = (F_1', F_2', \dots)$, and $\mathcal{F}'' = (F_1'', F_2'', \dots)$ which we need to "unify" slightly different distributions of outputs with respect to injections of invalid ciphertexts. A ciphertext is invalid if it decrypts to $\perp$ or if the first two bytes of its plaintext do not match the two-byte protocol version identifier. In both $\mathbf{R}$ and $\mathbf{S}$, if an invalid ciphertext is injected in a session with nonces η_C and η_{sid}, a random pre-master secret pmk^* is chosen and the random oracle (either RO_{384} or R) is evaluated on the value $pmk^*|\texttt{master secret}|\eta_C|\eta_{sid}$. Note that if the pre-master secret pmk^* collides with a value that was queried in combination with the same nonces η_C and η_{sid} before (either because it was computed by the client or because it was queried directly at RO_{384} by the attacker), the output of the random oracle, and hence at the server's interface, would of course be consistent with the previously obtained value. If the value pmk^* has not been used before, the response of the random oracle and hence the key output at the server's interface are uniformly random and independent of all previous values. The reduction $\mathbf{C}_{*,*,*}$, however, cannot distinguish invalid ciphertexts from ciphertexts decrypting to a pre-master secret that has not been used before (the PCA query to $\mathbf{G}^{\mathsf{nr\text{-}pca}}$ returns the same value on both), and hence always outputs a uniformly random key.

The pre-master secret pmk^* has 368 bits of entropy. Thus, after r different pre-master secrets have been queried at the random oracle with the same nonces, the probability for the freshly chosen pmk^* to collide with a previously used value is $r \cdot 2^{-368}$. With the remaining probability $1 - r \cdot 2^{-368}$, the query to the random oracle is fresh and the output distribution is uniform (i.e., every value has probability 2^{-384}). In case of $\mathbf{C}_{*,*,*}$, an invalid ciphertext always leads to a uniformly random output. As a result, an output of $\mathbf{R}$ or $\mathbf{S}$ on such a "dangerous" query with respect to an invalid ciphertext is slightly more likely to collide with previous outputs.

We rectify the above difference in probabilities by defining the three monotone binary outputs $\mathcal{F}$, $\mathcal{F}'$, and $\mathcal{F}''$ (for $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{C}_{*,*,*}$, respectively) as follows.

- The MBO $\mathcal{F}$ for $\mathbf{R}$ becomes 1 once in a session with nonces η_C and η_{sid}, an invalid ciphertext $\tilde{c}$ was input and lead to a query $pmk^*|\texttt{master secret}|\eta_C|\eta_{sid}$ (where pmk^* was chosen uniformly at random) to RO_{384}, and the same query was also asked at RO_{384} either by rsac or via the E-interface.
- The MBO $\mathcal{F}'$ for $\mathbf{S}$ becomes 1 once in a session with nonces η_C and η_{sid}, an invalid ciphertext $\tilde{c}$ was input and lead to a an evaluation on $pmk^*|\texttt{master secret}|\eta_C|\eta_{sid}$ (where

[12]Note that q' ranges over $1, 2, \dots$ independently of q as the MESs are defined for all reductions.
[13]Note that the event is well defined since for the RSA-PKCS scheme sk is uniquely defined by pk.

pmk^* was chosen uniformly at random) of R within σ, and the simulator σ computed an RSA ciphertext from $C = \rho^{-1}(\eta_C)$ (if defined) with the same pre-master secret pmk^* or R was also evaluated on the same value because of a query at the outside interface.

The above two MBO $\mathcal{F}$ (resp. $\mathcal{F}'$) guarantee that, given that the MBO remains 0 during a dangerous query, the output distribution is uniform. Using the value r defined above, this means that, for every possible key κ, the probability of the MBO remaining 0 *and* the key κ appearing is $(1 - r \cdot 2^{-368}) \cdot 2^{-384}$. What remains to be done is defining an MBO $\mathcal{F}''$ on $\mathbf{C}_{*,*,*}$ that leads to the same output distribution for such queries. But as the distribution in that case is uniform anyways, all that remains to be done is computing the probability for the MBO to be provoked during a "dangerous" query (see below), and provoking the MBO with the respective probability (independently of the value output by $\mathbf{C}_{*,*,*}$. The probability is computed in more detail as follows:

- if the query is the first query $pmk'|\texttt{master secret}|\eta|\tilde{\eta}$ for this value of pmk' to RO_{384}, where the first two bytes of pmk' match the two-byte protocol version identifier and an invalid ciphertext was sent to session $sid = (\eta, e)$ with nonce $\eta_{sid} = \tilde{\eta}$ before, then $\mathcal{F}'''$ becomes 1 with probability 2^{-368};
- if the query is sending an invalid ciphertext c in session (η, e), then the probability is computed by first counting the number r of *distinct* queries to RO_{384} and (potentially) the output at the corresponding client $\rho^{-1}(\eta)$ that collide in terms of generated keys; then $\mathcal{F}$ becomes 1 with probability $r \cdot 2^{-368}$.

Then define the MBOs $\mathcal{E}$ via $E_q \coloneqq \bigvee_{i,j} E_q^{i,j} \vee F_q$, $\mathcal{E}'$ via $E'_q \coloneqq \bigvee_{i,j} E_q^{i,j} \vee F'_q$, and $\mathcal{E}''$ via $E''_q \coloneqq \bigvee_{i,j} E_q^{i,j} \vee F''_q$.

By the definition of $\mathbf{C}_{i,j,q}$, $\mathcal{E}$, $\mathcal{E}'$, and $\mathcal{E}''$, one can verify that $\mathbf{R}^{\mathcal{E}} \overset{g}{\equiv} \mathbf{C}^{\mathcal{E}''}_{i,j,q}(\mathbf{G}^{\textsf{nr-pca}})^{-} \overset{g}{\equiv} \mathbf{S}^{\mathcal{E}'}$ for all pairs $(i,j,q) \in [|\mathcal{C}|] \times [n] \times \mathbb{N}$. Also, provoking the MBO $E_q^{i,j}$ in $\mathbf{C}_{i,j,q}(\mathbf{G}^{\textsf{nr-pca}})^{-}$ implies that the reduction $\mathbf{C}_{i,j}$ is successful in winning $\mathbf{G}^{\textsf{nr-pca}}$, so

$$\Gamma_q^{\mathbf{D}\mathbf{C}_{i,j,q}}(\mathbf{G}^{\textsf{nr-pca}}) = \Gamma_q^{\mathbf{D}}(\mathbf{C}_{i,j,q}\mathbf{G}^{\textsf{nr-pca}}) \geq \Gamma_q^{\mathbf{D}}\left(\mathbf{C}^{\mathcal{E}^{i,j}}_{i,j,q}(\mathbf{G}^{\textsf{nr-pca}})^{-}\right)$$

and $\sum_{i,j} \Gamma_q^{\mathbf{D}}\left(\mathbf{C}^{\mathcal{E}^{i,j}}_{i,j,q}(\mathbf{G}^{\textsf{nr-pca}})^{-}\right) + \Gamma_q^{\mathbf{D}}(\mathbf{C}^{\mathcal{F}''}_{i,j,q}(\mathbf{G}^{\textsf{nr-pca}})^{-}) \geq \Gamma_q^{\mathbf{D}}\left(\mathbf{C}^{\mathcal{E}''}_{*,*,q}(\mathbf{G}^{\textsf{nr-pca}})^{-}\right)$ by Lemma 20. The statement then follows using Lemma 19 and using the reduction $\mathbf{C}_q$ that chooses any one of the $\mathbf{C}_{i,j,q}$ with $i \in [|\mathcal{C}|]$ and $j \in [n]$ uniformly at random. The probability of provoking $\mathcal{F}$, $\mathcal{F}'$, resp. $\mathcal{F}''$ can be upper bounded by $q \cdot 2^{-368}$. $\square$

4 Expanding the Key

The master secret key is not used in the encryption and MAC schemes directly. The next protocol step, which we describe as a protocol $(\mathsf{expc}, \mathsf{exps})$, uses a HMAC-based PRF to generate sufficient key material (depending on the actual cipher suite) for two encryption and two MAC keys (one key per purpose and direction). Furthermore, the converters also generate the so-called "finished" messages which are used by the client and the server to confirm the computed keys.

More formally, starting from the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$, we expand the key using a pseudo-random function (PRF); our goal is to construct the ideal resource $\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!\Rightarrow\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}$. An advantage of this description is that it is the parallel composition of multiple "single-client" resources $\overset{KSP,*}{=\!\!\Rightarrow\!\bullet}_{\mathsf{cphs},n}$, which means all following protocol steps can be proven in a setting where there is only a single client, and then composed using the generic composition theorem.

The (extended) key space actually depends on the cipher suites in use. In this work, we do not specifically focus on any of the cipher suites, but rather define a function $\mathsf{cphs} : \{\mathsf{CAuth}, \mathsf{SAuth}, \mathsf{CEnc}, \mathsf{SEnc}, \mathsf{CIV}, \mathsf{SIV}\} \to \mathbb{Z}$, which outputs the length of each of the following keys: client_write_MAC_key ($\kappa_{C,a}$), server_write_MAC_key ($\kappa_{S,a}$), client_write_key ($\kappa_{C,e}$), server_write_key ($\kappa_{S,e}$), client_write_IV ($\kappa_{C,IV}$), server_write_IV ($\kappa_{S,IV}$). Note that the last two keys are often not generated, since they are only used for implicit nonce techniques, see [DR08]. Usually, the key length for each of the first four keys is 32 bytes, i.e. 256 bits. By convention, if $\kappa_{C,IV}, \kappa_{S,IV}$ are not generated, we write that $\mathsf{cphs}(\kappa_{C,IV}) = \mathsf{cphs}(\kappa_{S,IV}) = 0$. The set of all possible keys (parsed as a concatenation of all the aforementioned keys, in the given order) is denoted $\mathcal{K}$. The derived keys for each sessions are, in this sequence, client_write_MAC_key, server_write_MAC_key, client_write_key, server_write_key, client_write_IV, server_write_IV, and the two "finished" messages which are of length 96 bits each.

$$\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}$$

Initially, accept at the E-interface an input $b \in \{0,1\}$, and set $b_e = 0$ for $e \in [n]$. If $b = 0$, then output at both interfaces C and $S/1$ the same uniform random keys $\kappa_{C,a}, \kappa_{S,a}, \kappa_{C,e}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV}$ (with length indicated by cphs) and uniform $\xi_C, \xi_S \in \{0,1\}^{96}$, and provide bidirectional channels between those two interfaces.
Otherwise, if $b = 1$, then

- Upon input of the type $(\texttt{allow}, e)$ with $e \in [n]$ at the E-interface, if $b_e = 0$, draw uniformly random keys $\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV} \in \mathcal{K}$ of the lengths indicated by cphs, plus uniformly random strings $\xi_C, \xi_S \in \{0,1\}^{96}$. Output the tuple at interface C and set $b_e = 1$ (this ensures that the keys are only chosen once).

  On subsequent input $(\texttt{deliver})$, output the same tuple at interface S/e. Then, relay all communication between the interfaces C and E/C, and S/e and E/e.

- Upon input $(\texttt{inject}, e, \tilde{\kappa}_{C,a}, \tilde{\kappa}_{S,a}, \tilde{\kappa}_{C,e}, \tilde{\kappa}_{S,e}, \tilde{\kappa}_{C,IV}, \tilde{\kappa}_{S,IV}, \tilde{\xi}_C, \tilde{\xi}_S)$ with $e \in [n]$, if $b_e = 0$ then output the tuple at interface S/e and set $b_e = 1$. Then, relay all communication between the interfaces S/e and E/e.

In TLS the session keys are obtained via a PRF based on HMAC, taking as input the master secret value obtained in a previous step. Afterward, the client and server each generates a final message by again querying the PRF keyed with the master secret value, on input the hash of a concatenation of messages (basically the transcript of the session). This hash function, denoted H, is required to be collision resistant. For the purpose of our analysis, we will assume that the key material and the finished messages are derived by means of a pseudo-random function $PRF = (gen_{\mathrm{PRF}}, eval_{\mathrm{PRF}})$, with output length equal to $\max(|\mathsf{cphs}(\cdot)|, 96)$. For a more detailed discussion how this is achieved in TLS, we refer the reader to Appendix A.3.

The client converter, expc, behaves as follows:

1. Obtain the values $(\kappa, \eta_C, \eta_{sid}, m)$ from $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$, where m is a concatenation of messages.
2. Generate keys $(\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV}) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{key expansion}|\eta_C|\eta_{sid})$.
3. Generate messages $(\xi_C, \delta_C) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{client finished}|H(\eta_C|\eta_{sid}|m|\gamma))$ and $(\xi_S, \delta_S) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{server finished}|H(\eta_C|\eta_{sid}|m|c_{\xi_C}|\gamma))$.[14] In the above computation, the constant γ stands for the "ChangeCipherSpec" message, whereas the value c_{ξ_C} is computed as a function of ξ_C in a way that depends, as do the lengths of the computed

[14]The "extra" bits δ_C and δ_S are discarded.

keys, on the adopted cipher suite (this corresponds to the encryption of the finished message, computed in the record layer protocol).

4. Output $(\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV})$ and (ξ_C, ξ_S).

The server converter, exps behaves as follows. For each of the sessions described by a pair $sid = (\eta_C, e) \in \mathcal{N} \times [n]$:

1. Obtain the values (κ, η_{sid}, m) from $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$, where m is a concatenation of messages.
2. Generate keys $\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV} \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{key expansion}|\eta_C|\eta_{sid})$.
3. Generate "finished" messages $(\xi_C, \delta_C) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{client finished}|H(\eta_C|\eta_{sid}|m|\gamma))$ and $(\xi_S, \delta_S) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{server finished}|H(\eta_C|\eta_{sid}|m|c_{\xi_C}|\gamma))$.
4. Output $(\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV})$ and (ξ_C, ξ_S).

We aim for the statment that the protocol $(\mathsf{expc}, \mathsf{exps})$ constructs from $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ the resource $\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}$, i.e., the parallel composition of one copy of $\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}$ for each client $C \in \mathcal{C}$. This, however, is not directly true because in case $\prod_{C\in\mathcal{C}} \mathsf{expc}^C \mathsf{exps}^S$ $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ one can generate output at the server's sub-interface for client nonces $\eta \notin \rho(\mathcal{C})$, whereas with $\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}$ one cannot. We rectify this by introducing a "residual" resource $\tilde{\mathbf{R}}_{\mathsf{cphs},n}$ that is inactive in case $b = 0$ and behaves like $\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}$ at the S- and E-interfaces in case $b = 1$ ($\tilde{\mathbf{R}}_{\mathsf{cphs},n}$ does not have a C-interface). We then consider the system $[\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}$, i.e. the system where the S-interface is renamed as η-interface.

We show the following result, where we describe the pseudo-random function PRF by a converter prf and denote the system outputting a single 384-bit random string by $\mathbf{U}_{384}$ and the uniform random function (with the same output length as PRF) by $\mathbf{F}$.

Lemma 12. *The protocol* $(\mathsf{expc}, \mathsf{exps})$ *constructs from the* $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$*-resource the ideal resource* $\left[\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}, \bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})} [\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\right]$*, under the assumption that the hash function H is collision-resistant and the output of the pseudo-random function PRF is indistinguishable from a random function. More formally, there are reductions* $\mathbf{C}$ *and* $\mathbf{C}'$ *such that for all distinguishers* $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{expc}^C \mathsf{exps}^S \bot^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}, \bot^E \Big[\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}, \bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})} [\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\Big]\right) \leq \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}| \cdot \Delta^{\mathbf{DC}'}\left(\mathsf{prf}\,\mathbf{U}_{384}, \mathbf{F}\right), \tag{9}$$

and there is a simulator σ and reductions $\mathbf{C}''$ *and* $\mathbf{C}'''$ *such that for all distinguishers* $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{expc}^C \mathsf{exps}^S \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}, \sigma^E \Big[\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}, \bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})} [\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\Big]\right) \leq \Gamma^{\mathbf{DC}''}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}| \cdot \Delta^{\mathbf{DC}'''}\left(\mathsf{prf}\,\mathbf{U}_{384}, \mathbf{F}\right). \tag{10}$$

Proof. We first argue that condition (9) holds. Let $\mathbf{R}_\bot \coloneqq \prod_{C\in\mathcal{C}} \mathsf{expc}^C \mathsf{exps}^S \bot^E \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ and $\mathbf{S}_\bot \coloneqq \bot^E\left[\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!\!=\!\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}, \bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})} [\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\right]$. We notice that the main difference between the real and ideal system (when the cheating bit is set to $b = 0$), is that the latter chooses the keys and the finished messages (ξ_C, ξ_S) to be output at C and $S/1$ uniformly at random, whereas the former computes such values via the PRF.

We introduce a first hybrid systems $\mathbf{H}_\perp$, to deal with possible collisions in the hash function H. The system $\mathbf{H}_\perp$ is defined exactly as $\mathbf{R}_\perp$, with one difference: Whenever there exists two distinct tuples $(\eta_C, \eta_{sid}, m, c_{\xi_C})$ and $(\eta'_C, \eta'_{sid}, m', c'_{\xi'_C})$ such that either $(\eta_C, \eta_{sid}, \gamma)$ and $(\eta'_C, \eta'_{sid}, \gamma)$ or $(\eta_C, \eta_{sid}, m, c_{\xi_C}, \gamma)$ and $(\eta'_C, \eta'_{sid}, m', c'_{\xi'_C}, \gamma)$ are a collision for H, then the corresponding output of the hash function is re-sampled uniformly until a completely fresh (i.e., not previously used) value is found. We argue that a distinguisher between the two systems $\mathbf{R}$ and $\mathbf{H}_\perp$ can be used to build a reduction $\mathbf{C}$ breaking collision resistance of H. The reduction $\mathbf{C}$ connects with the inside interface to the game $\mathbf{G}^{\mathsf{CR}}$ and provides at the outside interface an emulation of $\mathbf{R}_\perp$, using a description of the hash function obtained from $\mathbf{G}^{\mathsf{CR}}$. A "collision" in the emulated execution can then be used to win the game $\mathbf{G}^{\mathsf{CR}}$ (for the exact definition of collision see the MBO below).

We define the monotone binary output (MBO) $\mathcal{E}$ on the systems $\mathbf{R}$, $\mathbf{H}_\perp$, and $\mathbf{C}\mathbf{G}^{\mathsf{CR}}$ as the following "collision" event: it becomes 1 once there are two distinct tuples $(\eta_C, \eta_{sid}, m, c_{\xi_C})$ and $(\eta'_C, \eta'_{sid}, m', c'_{\xi'_C})$ such that either $(\eta_C, \eta_{sid}, \gamma)$ and $(\eta'_C, \eta'_{sid}, \gamma)$ or $(\eta_C, \eta_{sid}, m, c_{\xi_C}, \gamma)$ and $(\eta'_C, \eta'_{sid}, m', c'_{\xi'_C}, \gamma)$ are a collision for H. Clearly the random systems $\mathbf{R}$ and $\mathbf{H}_\perp$ are equivalent conditioned on the MBO not being 1. Thus invoking Lemma 19 allows to conclude that $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{H}_\perp) \le \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{CR}})$.

Next, we argue that a distinguisher between $\mathbf{H}_\perp$ and $\mathbf{S}_\perp$ can be used together with a reduction $\mathbf{C}'$ to distinguish the pseudo-random function PRF from a truly random function. The reduction is actually a series of reductions $\mathbf{C}'_i$ for $i \in [|\mathcal{C}|]$, where the index pinpoints one client $C \in \mathcal{C}$. We assume some (e.g. lexicographic) order $\preceq$ over the set of clients, and write C_i to denote the i-th client with respect to this order. The reduction $\mathbf{C}'_i$ makes 3 queries to connected system and works as follows:

- For all the sessions between a client $C \succ C_i$, behave as in $\mathbf{H}_\perp$.
- For all the sessions between a client $C \prec C_i$, behave as in $\mathbf{S}_\perp$.
- In the session between C_i and $sid = (\eta, 1)$, forward $x_1 = \texttt{key expansion}|\eta|\eta_{sid}$, $x_2 = \texttt{client finished}|H(\eta|\eta_{sid}|m|\gamma)$ and $x_3 = \texttt{server finished}|H(\eta|\eta_{sid}|m|c_{x_2}|\gamma)$ to the connected system; note that the value c_{x_2} can be computed as a function of x_2 by only knowing the cipher suite in use. Emulate the session using the corresponding values y_1, y_2 and y_3, received from the inside interface. (In case a collision is found, similar to the MBO defined in $\mathbf{H}_\perp$, re-sample the output of the hash function uniformly.)

We note that if $\mathbf{C}'_i$ is connected to $\mathbf{F}$, then the values y_1, y_2 and y_3 are uniform, whereas if $\mathbf{C}'_i$ is connected to $\mathsf{prf}\,\mathbf{U}_{384}$, then they are computed as $eval_{\mathrm{PRF}}(\kappa, x)$ for $x \in \{x_1, x_2, x_3\}$ and a uniformly random key κ. Also, for all $0 \le i \le |\mathcal{C}|$, $\mathbf{C}'_i\,\mathsf{prf}\,\mathbf{U}_{384} \equiv \mathbf{C}'_{i+1}\mathbf{F}$. Furthermore, as $\mathbf{C}'_0 = \mathbf{H}_\perp\,\mathsf{prf}\,\mathbf{U}_{384}$ and $\mathbf{C}'_{|\mathcal{C}|}\,\mathbf{F} = \mathbf{S}_\perp$, the above argument concludes the proof of condition (9), where $\mathbf{C}'$ chooses any one of the $\mathbf{C}'_i$ uniformly at random.

To prove equation (10), consider the following simulator σ:

- Initially, σ sets $e_\eta = 1$ for all $\eta \in \mathcal{N}$. (The counters are kept consistent, i.e. they are increased whenever the simulator receives at the outside interface an $(\texttt{ack}, *)$ or $(\texttt{inject}, *)$ command.)
- Upon input $(\texttt{ack}, \eta)$ at the outside interface, if $e_\eta \le n$, then choose η_{sid} for $sid = (\eta, e_\eta)$ and output η_{sid} at the outside interface as being transmitted via $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$.

- Upon input $(\mathtt{allow}, C, \mathit{aux}, \tilde{\eta})$ at the outside interface, for $C \in \mathcal{C}$, issue $(\mathtt{allow}, e)$ at the C-sub-interface of $\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}$. Afterward, forward communication between the outside and the inside C sub-interfaces.

- Upon input $(\mathtt{deliver}, \eta, e, \mathit{aux})$, if $\eta \in \rho(\mathcal{C})$, $e < e_\eta$ and $(\mathtt{allow}, \rho^{-1}(\eta), *, \eta_{sid})$ was input at the outside interface before, issue $(\mathtt{deliver})$ at the $\rho^{-1}(\eta)$-sub-interface of the system $\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}$.

- Upon input $(\mathtt{inject}, \eta, e, \mathit{aux}, \kappa)$ at the outside interface, define $\mathit{sid} = (\eta, e)$ and compute

$$\begin{aligned}(\tilde{\kappa}_{C,a}, \tilde{\kappa}_{S,a}, \tilde{\kappa}_{C,e}, \tilde{\kappa}_{S,e}, \tilde{\kappa}_{C,IV}, \tilde{\kappa}_{S,IV}) &\leftarrow \mathit{eval}_{\mathrm{PRF}}(\kappa, \mathtt{key\ expansion}|\eta|\eta_{sid})\\ \tilde{\xi}_C &\leftarrow \mathit{eval}_{\mathrm{PRF}}(\kappa, \mathtt{client\ finished}|H(\eta|\eta_{sid}|\mathit{aux}|\gamma))\\ \tilde{\xi}_S &\leftarrow \mathit{eval}_{\mathrm{PRF}}(\kappa, \mathtt{server\ finished}|H(\eta|\eta_{sid}|\mathit{aux}|c_{\tilde{\xi}_C}|\gamma)).\end{aligned}$$

  Then issue $(\mathtt{inject}, e, \tilde{\kappa}_{C,a}, \tilde{\kappa}_{S,a}, \tilde{\kappa}_{C,e}, \tilde{\kappa}_{S,e}, \tilde{\kappa}_{C,IV}, \tilde{\kappa}_{S,IV}, \tilde{\xi}_C, \tilde{\xi}_S)$ at the inside $\rho^{-1}(\eta)$-sub-interface of $\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}$.

- After either a $(\mathtt{deliver}, \eta, e, *)$ or an $(\mathtt{inject}, \eta, e, *, *)$, forward communication between the outside and inside $\mathit{sid} = (\rho(\eta), e)$ sub-interfaces.

For the sake of brevity, we use the notation $\mathbf{R} \coloneqq \prod_{C\in\mathcal{C}} \mathsf{expc}^C \mathsf{exps}^S \mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ as well as $\mathbf{S} \coloneqq \sigma^E\Big[\bigotimes_{C\in\mathcal{C}} [\![\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))}, \bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})} [\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\Big]$. We notice that the only difference between $\mathbf{R}$ and $\mathbf{S}$ is within $(\mathtt{allow}, *)$ commands, as in the former the key material and the pair of values (ξ_C, ξ_S) are computed via the PRF and the hash function, whereas in the latter they are sampled uniformly. Similar to the availability proof, we consider a hybrid system $\mathbf{H}$ where we re-sample outputs of H corresponding to distinct inputs generating a collision (until a "fresh" value is found). Then one can describe a reduction $\mathbf{C}''$ that, together with a distinguisher telling apart $\mathbf{R}$ and $\mathbf{H}$, breaks the collision resistance of H. The description of the reduction is similar to the one considered in the availability case, and is therefore omitted.

Finally, we argue that a distinguisher between $\mathbf{H}$ and $\mathbf{S}$ can be used to build a reduction $\mathbf{C}'''$ breaking the pseudo-randomness of the PRF. We remark that there is a single key κ_C associated with client C, and the key at some interface $S/(\rho(C), e)$ is either the same key κ_C or it is an injected key. It follows that the the description of the reduction goes along the same lines to the one considered for the availability proof. Put together, the above arguments conclude the proof of equation (10). □

5 Constructing a Unilaterally Secure Channel

We describe the goal of the TLS *record layer* as constructing, from a unilateral key and insecure communication channels, a bidirectional unilaterally secure communication channel. TLS specifies several alternative cipher suites that are supposed to achieve this constructive step. The most widely used ones are based on an Authenticate-then-Encrypt combination of a MAC scheme and a symmetric encryption scheme, but [DR08] also specifies the possibility of using a monolithic authenticated encryption scheme. In this section, we prove the Authenticate-then-Encrypt modes based on [MT10], leaving the other cipher suites for future work.

As discussed in Section 1 and in previous work, the "finished" message of the TLS protocol cannot be regarded as part of the handshake if one wants to prove a strong security notion for the key. (The reason is that the actual key is used to protect the finished messages, which allows to verify whether an obtained key is correct.) Hence, as [JKSS12], we prove the record layer protocol *including* the finished messages and with the assumption of only a unilaterally authenticated key. The resource we want to construct by the record protocol is the "unilateral" channel $\twoheadleftarrow\overset{*}{-}\!\twoheadrightarrow\!\bullet_n$ described below.

$$\twoheadleftarrow\overset{*}{-}\!\twoheadrightarrow\!\bullet_n$$

Initially, accept at the E-interface an input $b \in \{0,1\}$.
If $b = 0$, then behave as a (secure multi-message) channel between interfaces C and $S/1$.
Otherwise, if $b = 1$, then:

- Upon the *first* input $(\texttt{allow}, e)$ with $e \in [n]$ at the E-interface (if e was not used before), provide a secure multiple-use (i.e., keep a buffer of undelivered messages) channel between C and S/e. In particular:
  - On input a message $m \in \{0,1\}^{\leq 16384}$ at the C-interface, output $|m|$ at interface E.
  - On input $(\texttt{deliver}, \texttt{client})$ at the E-interface, deliver the next message at S/e.
  - On input a message $m' \in \{0,1\}^{\leq 16384}$ at the S/e-interface, output $|m'|$ at interface E.
  - On input $(\texttt{deliver}, \texttt{server})$ at the E-interface, deliver the next message at C.
- After input $(\texttt{conquer}, e)$ with $e \in [n]$ at the E-interface (if e was not used before), forward messages in $\{0,1\}^{\leq 16384}$ bidirectionally between the S/e- and E/e-interfaces.

Generally, the record layer protocol is a pair of converters which both obtain at the respective inside interfaces keys and finished messages (as given by the $\overset{KSP,*}{=\!=\!\bullet}$-resource). The client's converter first sends the ξ_C-message (authenticated and encrypted), and then obtains, decrypts, and checks the ξ_S-message. If the check succeeds, payload messages are processed and transmitted. The server's converter first waits for the (encrypted) ξ_C-message, decrypts, and checks. If successful, the converter sends ξ_S authenticated and encrypted, and later processes and transmits payload messages. More precisely, the two converters are described as follows.

The client's converter behaves as follows.

1. Obtain at the inside interface keys $\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV}$ and "finished" messages ξ_C, ξ_S. Use the scheme(s) specified in the cipher suite with the keys $\kappa_{C,a}$ and $\kappa_{C,e}$, respectively, to process the message ξ_C, and send the obtained ciphertext via the inside interface.
2. Upon receiving a (ciphertext) message at the inside interface, process the message with the keys $\kappa_{S,e}$ and $\kappa_{S,a}$, respectively. Compare the plaintext to ξ_S. If any one of the above steps fails, abort.
3. Messages obtained at the outside interface are processed with the scheme(s) specified by the respective cipher suite using the keys $\kappa_{C,a}$ and $\kappa_{C,e}$ and sent via the inside interface. Further ciphertexts obtained at the inside interface are also processed with $\kappa_{S,e}$ and $\kappa_{S,a}$, the plaintexts are output at the outside interface. If any (MAC) verification fails, abort.

The server's converter behaves as follows:

1. Obtain at the inside interface keys $\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV}$ and "finished" messages ξ_C, ξ_S.
2. Upon receiving a message at the inside interface, process the message according to the specified cipher suite with the keys $\kappa_{C,e}$ and $\kappa_{C,a}$. Compare the plaintext to ξ_C. If any

one of the above steps fails, abort. Use the scheme(s) of the specified cipher suite with the keys $\kappa_{S,a}$ and $\kappa_{S,e}$ to process the message ξ_S, and send the obtained ciphertext via the inside interface.

3. Messages obtained at the outside interface are processed with the scheme(s) (using the keys $\kappa_{S,a}$ and $\kappa_{S,e}$) and sent via the inside interface. Further ciphertexts obtained at the inside interface are processed (with $\kappa_{C,e}$ and $\kappa_{C,a}$), the plaintexts are output at the outside interface. If any (MAC) verification fails, abort.

5.1 Cipher Suites Based on Stream Ciphers

The TLS standard [DR08] describes the record layer protocol based on stream ciphers in Section 6.2.3.1, the standard cipher suites using this type of encryption scheme are based on the cipher RC4. We prove the security following [MT10, Corollary 1] based on the assumption that RC4 produces a stream of pseudo-random bits.[15] In more detail, we formalize the assumption on RC4 by requiring that the distinguishing advantage between $\mathsf{rc4}\ \mathbf{U}_{128}$ (i.e., the stream generated by RC4 when initialized with a uniformly random 128-bit key) and $\mathbf{U}^*$ (a stream of uniformly random bits[16]) is small. The scheme is formalized as the pair $(\mathsf{atec}, \mathsf{ates})$ of converters.

Lemma 13. *The protocol* $(\mathsf{atec}, \mathsf{ates})$ *constructs from* $\overset{KSP,*}{=\!\Rightarrow\!\bullet}$ *the channel* $\twoheadleftarrow\!\overset{*}{-}\!\twoheadrightarrow\!\bullet_n$*, under the assumptions that RC4 is pseudo-random and HMAC is strongly unforgeable. More formally,*

$$\mathsf{atec}^C \mathsf{ates}^S \bot^E \overset{KSP,*}{=\!\Rightarrow\!\bullet} \quad \equiv \quad \bot^E \twoheadleftarrow\!\overset{*}{-}\!\twoheadrightarrow\!\bullet_n, \tag{11}$$

and there are a simulator σ *and reductions* $\mathbf{C}$, $\mathbf{C}'$ *such that for each distinguisher* $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\mathsf{atec}^C \mathsf{ates}^S \overset{KSP,*}{=\!\Rightarrow\!\bullet}, \sigma^E \twoheadleftarrow\!\overset{*}{-}\!\twoheadrightarrow\!\bullet_n\right) \leq 2 \cdot \Delta^{\mathbf{DC}}\left(\mathsf{rc4}\ \mathbf{U}_{128}, \mathbf{U}^*\right) + 2 \cdot \Gamma^{\mathbf{DC}'}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right). \tag{12}$$

Proof sketch. Equation (11) follows because the resource $\overset{KSP,*}{=\!\Rightarrow\!\bullet}$ outputs the same keys and "finished" messages at interfaces C and S, so the verification of these messages succeeds as atec and ates compute the same key streams and MACs.

To prove the condition in equation (12), we describe a simulator σ that initially sets the bit $b = 1$ at the inside interface. The simulator initializes bits $b_e = 0$ for each $e \in [n]$. Then:

- Upon input $(\mathtt{allow}, e)$ at the outside interface, if $e \in [n]$ and $b_e = 0$, then set $b_e = 1$ and $\bar{e} = e$. Simulate the transmission of the client's finished message (a uniform random string of length 256 bits—96 bits "finished" message and 160 bits MAC) as the first message c_1 from C to S/e.
- Once both $(\mathtt{deliver})$ has been input at the outside interface and the first message $\tilde{c}_1$ is delivered to S in session $\bar{e}$, if $c_1 = \tilde{c}_1$ then simulate a finished message from the server to the client, again by choosing a 256-bit string c_2 uniformly at random.
  In the following, upon input the ith message length ℓ_i corresponding to a server message in session $\bar{e}$ at the inside interface, output a uniformly random string of length $\ell_i + 160$. Also, whenever a message is delivered (via the outer interface) to the server session $\bar{e}$, check whether *exactly* the same messages were simulated as being sent by the client before. In this case, input $(\mathtt{ack}, C)$ at the inside interface, otherwise halt the server session $\bar{e}$ (i.e., stop processing messages for this server session, and, in real implementations, send an alert; however, for our treatment we omit alerts from the protocol description.).

[15] As demonstrated in [ABP+13], the assumption that RC4 is pseudo-random is dangerous. The proof extends to other stream ciphers.

[16] Formally, $\mathbf{U}^*$ and $\mathsf{rc4}$ allow to obtain the stream by querying for one bit at a time.

- Once the first message $\tilde{c}_2$ is delivered to C, if $c_2 = \tilde{c}_2$ then record the client as active. In the following, upon input the ith message length ℓ_i corresponding to a client message at the inside interface, output a uniformly random string of length $\ell_i + 160$. Also, as above, whenever a message is delivered to the client, if *exactly* the same sequence of message was simulated as being sent by the server's session $\bar{e}$ before, then input $(\texttt{ack}, S)$ at the inside interface, otherwise halt the client (i.e., stop processing messages for the client—note that we simplify the protocol and do not handle error messages).
- Upon input $(\texttt{inject}, e, \bar{\kappa}_{C,a}, \bar{\kappa}_{C,e}, \bar{\kappa}_{S,a}, \bar{\kappa}_{S,e}, \bar{\kappa}_{C,IV}, \bar{\kappa}_{S,IV}, \bar{\xi}_C, \bar{\xi}_S)$ at the E-interface with $e \in [n]$ and $b_e = 0$, set $b_e = 1$ and record the values. When the first message is delivered to the session e, check whether the message is a correctly MAC'ed and encrypted version (with $\bar{\kappa}_{C,a}$ and $\bar{\kappa}_{C,e}$) of $\bar{\xi}_C$ (if not, abort the server session e). Respond with a correctly MAC'ed and encrypted version (with $\bar{\kappa}_{S,a}$ and $\bar{\kappa}_{S,e}$) of ξ_S.
  Subsequently, MAC and encrypt messages sent in the server session e with the keys $\bar{\kappa}_{S,a}$ and $\bar{\kappa}_{S,e}$. For messages given at the outside interface for this session, decrypt with $\bar{\kappa}_{C,e}$ and verify the MAC with $\kappa_{C,a}$. In case of success, inject the resulting message via the inside interface, otherwise halt the server session e.

First, we note that the simulation of all sessions except for $\bar{e}$ is perfect, as the simulator makes exactly the same computations as the protocol.

To prove the security statement, we use a hybrid system $\mathbf{H}_1$ similarly to $\sigma^E \twoheadleftarrow\overset{*}{\relbar}\twoheadrightarrow\bullet_n$ with the difference that the key stream is generated by RC4 with a uniformly random key instead of uniformly at random. The reduction system $\mathbf{C}$ simulates all sessions similarly to $\sigma^E \twoheadleftarrow\overset{*}{\relbar}\twoheadrightarrow\bullet_n$, but in session $\bar{e}$ it uses the key stream from the connected system (with probability $\frac{1}{2}$ it does so for the client while using a fully random stream for the server, with the remaining probability it uses the given stream for the client and generates the server's stream using RC4). This means that $\Delta^{\mathbf{D}}\left(\mathsf{atec}^C \mathsf{ates}^S \overset{KSP,*}{=\!\!=\!\!\bullet}, \mathbf{H}_1\right) \leq 2 \cdot \Delta^{\mathbf{DC}}(\mathsf{rc4}\ \mathbf{U}_{128}, \mathbf{U}^*)$. Then, we use [MT10, Corollary 1] twice, once for each direction, to obtain the statement $\Delta^{\mathbf{D}}\left(\mathbf{H}_1, \sigma^E \twoheadleftarrow\overset{*}{\relbar}\twoheadrightarrow\bullet_n\right) \leq 2 \cdot \Gamma^{\mathbf{DC}'}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right)$ and apply the triangular inequality to conclude. □

5.2 Cipher Suites Based on CBC Encryption

The TLS standard [DR08] describes the record layer protocol based on CBC encryption in Section 6.2.3.2, the standard cipher suites using this encryption mode are based on either 3DES or AES. We prove the security following [MT10, Corollary 2], based on the assumption that the used block cipher is a (super[17]) PRP. In more detail, we formalize the assumptions on the block ciphers by requiring that the distinguishing advantage between $\mathsf{bc}\ \mathbf{U}_k$ (i.e., the block cipher bc with block length ℓ which might e.g. be 3DES or AES initialized with a uniformly random key of appropriate length) and $\mathbf{P}_\ell$ (a uniformly random permutation) is small, even allowing both forward and backward queries.

In the following lemma, we use the converters atec' and ates' that implement the Authenticate-then-Encrypt composition of a CBC-mode with block cipher bc and a strongly unforgeable MAC. As we base the proof on [MT10, Corollary 2], it only applies to the case where the padding used by TLS is unique in the sense that it is the shortest possible such padding (and no length-hiding techniques are used).

Lemma 14. *The protocol* $(\mathsf{atec}', \mathsf{ates}')$ *constructs from* $\overset{KSP,*}{=\!\!=\!\!\bullet}$ *the channel* $\twoheadleftarrow\overset{*}{\relbar}\twoheadrightarrow\bullet_n$*, under the*

[17]In the reduction, it is necessary to make *inverse* queries to the permutation. This is unclear in [MT10].

assumptions that bc *is a (super) PRP and HMAC is strongly unforgeable. More formally,*

$$\mathsf{atec}'^{C}\mathsf{ates}'^{S}\bot^{E} \overset{KSP,*}{=\!\!\Rightarrow\!\bullet} \quad\equiv\quad \bot^{E} \twoheadleftarrow\overset{*}{-}\!\!\twoheadrightarrow\bullet_{n}, \tag{13}$$

and there are a simulator σ and reductions $\mathbf{C}$, $\mathbf{C}'$ such that for each distinguisher $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\mathsf{atec}'^{C}\mathsf{ates}'^{S} \overset{KSP,*}{=\!\!\Rightarrow\!\bullet}, \sigma^{E} \twoheadleftarrow\overset{*}{-}\!\!\twoheadrightarrow\bullet_{n}\right) \le 2\cdot\Delta^{\mathbf{DC}}\left(\mathsf{bc}\ \mathbf{U}_k, \mathbf{P}_\ell\right) + 2\cdot\Gamma^{\mathbf{DC}'}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right) + \frac{(ql)^2}{2^{\ell-1}}, \tag{14}$$

where q is the number of messages sent and l is the length (in blocks) of the longest message.

Proof sketch. The availability condition follows as in Lemma 13.

To prove the condition in equation (14), we describe a simulator σ that initially sets the bit $b = 1$ at the inside interface. The simulator initializes bits $b_e = 0$ for each $e \in [n]$. Then:

- Upon input $(\texttt{allow}, e)$ at the outside interface, if $e \in [n]$ and $b_e = 0$, then set $b_e = 1$ and $\bar{e} = e$. Simulate the transmission of the client's finished message (a uniform random string of length $\ell \cdot \lceil \frac{256}{\ell} \rceil$ bits—96 bits "finished" message and 160 bits MAC, padded to the next block size) as the first message c_1 from C to S/e.
- Once both $(\texttt{deliver})$ has been input at the outside interface and the first message $\tilde{c}_1$ is delivered to S in session $\bar{e}$, if $c_1 = \tilde{c}_1$ then simulate a finished message from the server to the client, again by choosing a bit string c_2 of appropriate length uniformly at random.
  In the following, upon input the ith message length ℓ_i corresponding to a server message in session $\bar{e}$ at the inside interface, output a uniformly random string of length $\ell \cdot \left\lceil \frac{\ell_i + 160}{\ell} \right\rceil$.
  Whenever a message is delivered (via the outer interface) to the server session $\bar{e}$, behave as the simulator in Lemma 13.
- Once the first message $\tilde{c}_2$ is delivered to C, if $c_2 = \tilde{c}_2$ then record the client as active.
  In the following, upon input the ith message length ℓ_i corresponding to a client message at the inside interface, output a uniformly random string of length $\ell \cdot \left\lceil \frac{\ell_i + 160}{\ell} \right\rceil$. Also, as above, whenever a message is delivered to the client, behave as the simulator in Lemma 13.
- Upon input $(\texttt{inject}, e, \bar{\kappa}_{C,a}, \bar{\kappa}_{C,e}, \bar{\kappa}_{S,a}, \bar{\kappa}_{S,e}, \bar{\kappa}_{C,IV}, \bar{\kappa}_{S,IV}, \bar{\xi}_C, \bar{\xi}_S)$ at the E-interface with $e \in [n]$ and $b_e = 0$, set $b_e = 1$ and record the values. When the first message is delivered to the session e, check whether the message is a correctly MAC'ed, padded, and encrypted version (with $\bar{\kappa}_{C,a}$ and $\bar{\kappa}_{C,e}$) of $\bar{\xi}_C$ (if not, abort the server session e). Respond with a correctly MAC'ed, padded, and encrypted version (with $\bar{\kappa}_{S,a}$ and $\bar{\kappa}_{S,e}$) of ξ_S.
  Subsequently, MAC, pad, and encrypt messages sent in the server session e with the keys $\bar{\kappa}_{S,a}$ and $\bar{\kappa}_{S,e}$. For messages given at the outside interface for this session, decrypt with $\bar{\kappa}_{C,e}$ and verify the padding and the MAC with $\kappa_{C,a}$. In case of success, inject the resulting message via the inside interface, otherwise halt the server session e.

First, we note that the simulation of all sessions except for $\bar{e}$ is perfect, as the simulator makes exactly the same computations as the protocol.

To prove the security statement, we use a hybrid system $\mathbf{H}_1$ similarly to $\sigma^{E} \twoheadleftarrow\overset{*}{-}\!\!\twoheadrightarrow\bullet_{n}$ with the difference that the CBC scheme is computed using a uniformly random permutation $\mathbf{P}_\ell$ (instead of bc). The reduction system $\mathbf{C}$ simulates all sessions similarly to $\sigma^{E} \twoheadleftarrow\overset{*}{-}\!\!\twoheadrightarrow\bullet_{n}$, but in session $\bar{e}$ it uses the connected system (with probability $\frac{1}{2}$ it does so for the client while using a fully random permutation for the server, with the remaining probability it uses the given permutation for the server and generates the server's stream using bc). This means that $\Delta^{\mathbf{D}}\left(\mathsf{atec}'^{C}\mathsf{ates}'^{S} \overset{KSP,*}{=\!\!\Rightarrow\!\bullet}, \mathbf{H}_1\right) \le 2\cdot\Delta^{\mathbf{DC}}\left(\mathsf{bc}\ \mathbf{U}_k, \mathbf{P}_\ell\right)$. Then, we use [MT10, Corollary 2] twice, once for each direction, to obtain the statement $\Delta^{\mathbf{D}}\left(\mathbf{H}_1, \sigma^{E} \twoheadleftarrow\overset{*}{-}\!\!\twoheadrightarrow\bullet_{n}\right) \le 2\cdot\Gamma^{\mathbf{DC}'}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right) + \frac{(ql)^2}{2^{\ell-1}}$ and apply the triangular inequality to conclude. □

6 Reconstructing TLS

In this section we argue that the composition of the converters we presented in Sections 3.3.2, 4, and 5 forms in fact the TLS protocol, and then give the full security statements for each of the cipher suites.

Note that in our deconstruction of TLS, we use an intermediate converter that does not appear in the TLS-DH and TLS-RSA versions, namely constructing the authenticated network resource $\rhd\!\!-\!\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$. As the difference between the TLS-DH/TLS-RSA and the TLS-DHE protocols only appears in the construction of the master secret resource (in our de-construction, obtaining the master key resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ from the nonce-exchange resource $\mathsf{SNET}_{\mathsf{N},\rho,n}$), we only consider the TLS-DHE variant, noting that the same considerations hold for the TLS-DH/TLS-RSA versions.

In fact, we describe TLS as a composition of a client TLS-DHE converter (denoted as $\mathsf{tlsdhec}$) and one server TLS-DHE converter. The client TLS-DHE converter $\mathsf{tlsdhec}$ represents the composition of all the converters mounted at the client interfaces of our respective resources, i.e., $\mathsf{tlsdhec} = \mathsf{atec} \circ \mathsf{expc} \circ \mathsf{dhec} \circ \mathsf{vrf} \circ \mathsf{hec}$. We explicitly outline the resulting $\mathsf{tlsdhec}$ converter next:

1. hec: Obtain a (random) nonce $\eta_C \in \mathcal{N}$ (at the inside interface) and send it to the server via the inside interface. Upon receiving nonce $\tilde{\eta}$ at the inside interface, output $(\eta_C, \tilde{\eta})$ at the outside interface.
2. vrf: Upon receiving message *cert* at the inside interface corresponding to $\mathsf{SNET}_{\mathsf{N},\rho,n}$, query $(\mathtt{verify}, \mathit{cert})$ at the inside sub-interface corresponding to $\mathsf{PKI}_{\mathfrak{F}}$; abort if the verification fails or if *cert* is not a well-formed certificate $\mathit{cert} = (\mathit{vk}, f(\mathit{vk}))$. Upon obtaining a second message m' from $\mathsf{SNET}_{\mathsf{N},\rho,n}$, parse m' as (m, s) (abort if that is impossible). If $\mathit{vrf}(\eta_C|\eta_{sid}|m, s; \mathit{vk}) = 1$, then output $(\mathit{cert}, \eta_C, \eta_{sid}, m, s)$ at the outside interface. (Otherwise abort.)
3. dhec: Parse message m (obtained at the inside interface) as $p|g|g' = m$ (abort it impossible). Choose $u \leftarrow\!\!\$ \{1, \ldots, q\}$ (with $q = |\mathbf{Z}_p^\times|$) and input g^u at the inside interface. Query ${g'}^u|\mathtt{master\ secret}|\eta_C|\eta_{sid}$ at RO_{384}, in order to obtain a key $\kappa \in \{0,1\}^{384}$. Output $(\kappa, \eta_C, \eta_{sid}, \mathit{aux}|m|s|g^u)$.
4. expc: Use the value κ to generate keys
$$(\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV}) \leftarrow \mathit{eval}_{\mathrm{PRF}}(\kappa, \mathtt{key\ expansion}|\eta_C|\eta_{sid}).$$
Using the concatenation of the previously transmitted messages $m \coloneqq \mathit{cert}|p|g|g'|s$, compute the "finished" messages $(\xi_C, \delta_C) \leftarrow \mathit{eval}_{\mathrm{PRF}}(\kappa, \mathtt{client\ finished}|H(\eta_C|\eta_{sid}|m|\gamma))$ and $(\xi_S, \delta_S) \leftarrow \mathit{eval}_{\mathrm{PRF}}(\kappa, \mathtt{server\ finished}|H(\eta_C|\eta_{sid}|m|c_{\xi_C}|\gamma))$. In the above computation, the constant γ stands for the "ChangeCipherSpec" message, whereas the value c_{ξ_C} is computed as a function of ξ_C depending on the adopted cipher suite (in particular, ξ_C needs to be included in the hash of the message sent by the server). Output $(\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV})$ and (ξ_C, ξ_S).
5. atec: Use the record layer scheme(s) specified by the cipher suite with the keys $\kappa_{C,a}$ and $\kappa_{C,e}$ obtained at the inside interface, respectively, to process the message ξ_C (obtained with the keys), and to computed and send the obtained ciphertext via the inside interface. Upon receiving a message at the inside interface, process it using the keys $\kappa_{S,e}$ and $\kappa_{S,a}$. Compare the plaintext to ξ_S. If any one of the above steps fails, abort. From this point on, messages obtained at the outside interface are processed with the specified scheme(s) using keys $\kappa_{C,a}$ and $\kappa_{C,e}$ and sent via the inside interface. Further ciphertexts obtained at the inside interface are processed with the keys $\kappa_{S,e}$ and $\kappa_{S,a}$, the plaintexts are output at the outside interface. If any (MAC) verification fails, abort.

Note that the composite converter corresponds exactly to the client's protocol in TLS, with the specification that, in the expansion step, the client first computes ξ_S as the PRF, under the key κ of the hash of all the past messages, including ξ_C. We note that our description of the protocol omits several constants that appear in the original protocol.

The server converter $\mathsf{tlsdhes}$ connects to the S-interface, and is composed of all the converters connected to the S-interface, i.e., $\mathsf{tlsdhes} = \mathsf{ates} \circ \mathsf{exps} \circ \mathsf{dhes}_{\mathcal{G}} \circ \mathsf{sgn} \circ \mathsf{hes}$. We explicitly outline the resulting $\mathsf{tlsdhes}$ converter below:

1. hes: Upon receiving a nonce $\tilde{\eta_C}$ at the inside C-sub-interface for some $C \in \mathcal{A}_{\mathrm{TCP}}$, choose a nonce $\eta_{sid} \leftarrow\!\!\$\ \mathsf{N}$, send η_{sid} via the inside C-sub-interface. Output η_{sid} at outside sub-interface $\tilde{sid}$.
2. sgn: Initially compute $(sk, vk) \leftarrow gen$. Input vk at the inside sub-interface corresponding to $\mathsf{PKI}_{\mathfrak{F}}$, obtaining a response s, and set $cert = (vk, s)$. Output $cert$ at the outside interface (as auxiliary information). Subsequently, for each inside sub-interface $sid = (\eta, e)$ outputting a nonce output η_{sid}, output η_{sid} at the respective outside sub-interface and send $cert$ via $\mathsf{SNET}_{\mathsf{N},\rho,n}$ (in the respective session). Obtaining a message m at the outside (sub-interface for session sid), compute $s \leftarrow sign(\eta_C|\eta_{sid}|m, sk)$ and send (m, s) in session sid. Output s at the respective outside sub-interface.
3. $\mathsf{dhes}_{\mathcal{G}}$: Upon obtaining a nonce η_{sid} at the inside sub-interface $sid = (\eta_C, e)$, choose a modulus $p \in \mathbb{N}$ and a generator $g \in \mathbf{Z}_p^\times$ according to the distribution $\mathcal{G}$. Also, choose an exponent $v \leftarrow\!\!\$\ \{1, \ldots, |\mathbf{Z}_p^\times|\}$. Input the value $m = p|g|g^v$ at the respective inside sub-interface (obtaining a signature s in response). Upon receiving a group element $\tilde{g}$ at the respective inside sub-interface, query $\tilde{g}^v|\texttt{master secret}|\eta_C|\eta_{sid}$ at RO_{384}, call the result κ. Output $(\kappa, \eta_{sid}, aux|m|s|\tilde{g})$ at the respective outside sub-interface.
4. exps: Obtaining $(\kappa, \eta_{sid}, aux|m|s|\tilde{g})$ at the inside sub-interface of session $sid = (\eta_C, e)$, generate keys $\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV} \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{key expansion}|\eta_C|\eta_{sid})$. Denote $m \coloneqq cert|p|g|g'|s$. Generate messages $(\xi_C, \delta_C) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{client finished}| H(\eta_C|\eta_{sid}|m|\gamma))$ and $(\xi_S, \delta_S) \leftarrow eval_{\mathrm{PRF}}(\kappa, \texttt{server finished}|H(\eta_C|\eta_{sid}|m|c_{\xi_C}|\gamma))$. Output $(\kappa_{C,a}, \kappa_{C,e}, \kappa_{S,a}, \kappa_{S,e}, \kappa_{C,IV}, \kappa_{S,IV})$ and (ξ_C, ξ_S) at the outside sub-interface sid.
5. ates: There is one such converter for each nonce $\eta \in \mathcal{N}$, which connects at the respective sub-interface. Upon receiving a message at the inside interface, process the message with the scheme(s) specified by the cipher suite using the keys $\kappa_{C,e}$ and $\kappa_{C,a}$. Compare the plaintext to ξ_C. If any one of the above steps fails, abort. Use again the specified schemes, now using the keys $\kappa_{S,a}$ and $\kappa_{S,e}$, to process the message ξ_S, and send the obtained ciphertext via the inside interface. Messages obtained at the outside interface are processed with the specified scheme(s) using the keys $\kappa_{S,a}$ and $\kappa_{S,e}$ and sent via the inside interface. Further ciphertexts obtained at the inside interface are processed using $\kappa_{C,e}$ and $\kappa_{C,a}$, the plaintexts are output at the outside interface. If any (MAC) verification fails, abort.

Note once more that this amounts to the server protocol in TLS-DHE. In the following section we give the full security statements for all versions TLS-DH, TLS-DHE, and TLS-RSA.

6.1 Full Security Statements

Summarizing the bounds obtained in the previous sections, the full security statements for TLS-DH, TLS-DHE, and TLS-RSA are as follows. We write the theorems for the cipher suites based on the stream cipher RC4 only, but the analogous theorems for the CBC-based cipher suites are obtained in the same way.

Similarly to Section 4, need to introduce a “residual” resource $\hat{\mathbf{R}}_n$ that is inactive in case $b = 0$ and behaves like $\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet_n$ at the S- and E-interfaces in case $b = 1$ ($\hat{\mathbf{R}}_n$ does not have a C-interface). We then consider, for each nonce $\eta \notin \rho(\mathcal{C})$, the system $[\![\hat{\mathbf{R}}_n]\!]^{(S/\eta)}$, i.e. the system where the S-interface becomes the η-sub-interface of the S-interface. For brevity, we define $\hat{\mathbf{R}}_n^{\rho(\mathcal{C})} :\!= \bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})}[\![\hat{\mathbf{R}}_n]\!]^{(S/\eta)}$.

We start by showing the complete security statement for a cipher suite based on TLS-DH and a stream cipher, here RC4.

Theorem 15 (TLS-DH). *Let $\mathcal{C} \subseteq \mathcal{A}_{\mathrm{TCP}}$ be a set of clients. The TLS-DH protocol constructs, for each client $C \in \mathcal{C}$, one unilaterally secure channel $\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet_n$ from NET, PKI, and RO_{384}. Concretely, there exist reductions $\mathbf{C}$ and $\mathbf{C}'$ such that:*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}}\mathsf{tlsdhc}^{C}\mathsf{tlsdhs}_{\mathcal{G},n}{}^{S}\bot^{E}\left[\mathsf{NET},\mathsf{PKI},\mathsf{RO}_{384}\right],\bot^{E}\left[\bigotimes_{C\in\mathcal{C}}[\![\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet_n]\!]^{(C,S/\rho(C))},\hat{\mathbf{R}}_n^{\rho(\mathcal{C})}\right]\right)$$
$$\leq \binom{|\mathcal{C}|}{2}\cdot 2^{-224} + \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}'}\left(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F}\right),$$

and there are a simulator σ and reductions $\mathbf{C}$, $\mathbf{C}'$, $\mathbf{C}''$, $\mathbf{C}'''$, and $\mathbf{C}^{(iv)}$ such that for each distinguisher $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}}\mathsf{tlsdhc}^{C}\mathsf{tlsdhs}_{\mathcal{G},n}{}^{S}\left[\mathsf{NET},\mathsf{PKI},\mathsf{RO}_{384}\right],\sigma^{E}\left[\bigotimes_{C\in\mathcal{C}}[\![\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet_n]\!]^{(C,S/\rho(C))},\hat{\mathbf{R}}_n^{\rho(\mathcal{C})}\right]\right)$$
$$\leq \left(\binom{n}{2}+\binom{|\mathcal{C}|}{2}\right)\cdot 2^{-224} + \Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) + \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}''}\left(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F}\right)$$
$$+\,|\mathcal{C}|\left(2\cdot\Delta^{\mathbf{DC}'''}\left(\mathsf{rc4}\ \mathbf{U}_{128},\mathbf{U}^{*}\right) + 2\cdot\Gamma^{\mathbf{DC}^{(iv)}}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right)\right).$$

Proof. For the availability condition, we lose a term $\binom{|\mathcal{C}|}{2}2^{-224}$ in the construction of the resource NAME_ρ from scratch (see Section 3.2.1), and another term $\Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{CR}})+|\mathcal{C}|\cdot\Delta^{\mathbf{DC}'}(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F})$ in the construction of $\left[\bigotimes_{C\in\mathcal{C}}[\![\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))},\bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})}[\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\right]$ via $(\mathsf{expc},\mathsf{exps})$ (see Lemma 12). The converters ates connected to the interfaces corresponding to $\eta \in \mathcal{N}\setminus\rho(\mathcal{C})$ do not obtain keys and hence remain inactive.

For the security condition, we again lose a term $\binom{|\mathcal{C}|}{2}2^{-224}$ when constructing the resource NAME_ρ from scratch. We also lose: a term $\binom{n}{2}2^{-224}$ in the construction of $\mathsf{SNET}_{\mathsf{N},\rho,n}$ from the resources NET and NAME_ρ (see Lemma 7); a term $\Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}})$ in the construction of the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ from the resource $[\mathsf{RO}_{384},\mathsf{SNET}_{\mathsf{N},\rho,n},\mathsf{PKI}_{\mathfrak{F}}]$ (see Lemma 8); a term $\Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}''}(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F})$ in constructing the parallel composition of keys $\left[\bigotimes_{C\in\mathcal{C}}[\![\overset{KSP,*}{=\!=\!\bullet}_{\mathsf{cphs},n}]\!]^{(C,S/\rho(C))},\bigotimes_{\eta\in\mathcal{N}\setminus\rho(\mathcal{C})}[\![\tilde{\mathbf{R}}_{\mathsf{cphs},n}]\!]^{(S/\eta)}\right]$ via $(\mathsf{expc},\mathsf{exps})$ (see Lemma 12); finally, noting that we lose a term $2\cdot\Delta^{\mathbf{DC}'''}(\mathsf{rc4}\ \mathbf{U}_{128},\mathbf{U}^{*}) + 2\cdot\Gamma^{\mathbf{DC}^{(iv)}}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right)$ for each obtained channel between a client and server. The $|\mathcal{C}|$-many such channels are obtained by parallel composition. □

We obtain the following analogous statement for a cipher suite based on TLS-DHE together with a stream cipher.

Theorem 16 (TLS-DHE). *Let $\mathcal{C} \subseteq \mathcal{A}_{\mathrm{TCP}}$ be a set of clients. The TLS-DHE protocol constructs, for each client $C \in \mathcal{C}$, one unilaterally secure channel $\twoheadleftarrow\overset{*}{-}\twoheadrightarrow\bullet_n$ from NET, PKI, and RO_{384}.*

Concretely, there exist reductions $\mathbf{C}$ *and* $\mathbf{C}'$ *such that:*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{tlsdhec}^{C}\mathsf{tlsdhes}_{\mathcal{G},n}{}^{S}\bot^{E}\left[\mathsf{NET},\mathsf{PKI},\mathsf{RO}_{384}\right], \bot^{E}\left[\bigotimes_{C\in\mathcal{C}}[\![\leftarrow\!\!\overset{*}{-}\!\!\twoheadrightarrow\!\bullet_n]\!]^{(C,S/\rho(C))}, \hat{\mathbf{R}}_n^{\rho(\mathcal{C})}\right]\right)$$
$$\leq \binom{|\mathcal{C}|}{2}\cdot 2^{-224} + \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}'}\left(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F}\right),$$

and there are a simulator σ *and reductions* $\mathbf{C}$*,* $\mathbf{C}'$*,* $\mathbf{C}''$*,* $\mathbf{C}'''$*,* $\mathbf{C}^{(iv)}$*, and* $\mathbf{C}^{(v)}$ *such that for each distinguisher* $\mathbf{D}$*,*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{tlsdhec}^{C}\mathsf{tlsdhes}_{\mathcal{G},n}{}^{S}\left[\mathsf{NET},\mathsf{PKI},\mathsf{RO}_{384}\right], \sigma^{E}\left[\bigotimes_{C\in\mathcal{C}}[\![\leftarrow\!\!\overset{*}{-}\!\!\twoheadrightarrow\!\bullet_n]\!]^{(C,S/\rho(C))}, \hat{\mathbf{R}}_n^{\rho(\mathcal{C})}\right]\right)$$
$$\leq 2\cdot\left(\binom{n}{2}+\binom{|\mathcal{C}|}{2}\right)\cdot 2^{-224} + \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{uf\text{-}cma}}) + n\cdot|\mathcal{C}|\cdot\Gamma^{\mathbf{DC}'}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}})$$
$$+\,\Gamma^{\mathbf{DC}''}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}'''}\left(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F}\right)$$
$$+\,|\mathcal{C}|\cdot\left(2\cdot\Delta^{\mathbf{DC}^{(iv)}}\left(\mathsf{rc4}\ \mathbf{U}_{128},\mathbf{U}^{*}\right) + 2\cdot\Gamma^{\mathbf{DC}^{(v)}}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right)\right).$$

Proof. The proof of the availability condition follows exactly the same scheme as above. A similar argument holds for the security condition, with the exception that we lose a term $\Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{uf\text{-}cma}})$ while constructing the resource $\succ\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$ from the resources $\mathsf{SNET}_{\mathsf{N},\rho,n}$ and $\mathsf{PKI}_{\mathfrak{F}}$ (see Lemma 9), and then a term $n\cdot|\mathcal{C}|\cdot\Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}})$ in constructing the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ from the resources RO_{384} and $\succ\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$ (see Lemma 10). The remainder of the proof is as above. □

The analogous result also holds with respect to a cipher suite based on TLS-RSA together with a stream cipher. Note that the entire security statement is implicitly parametrized by *gen*, the RSA key generation algorithm, which is specialized for the considered key length. In particular, this affects the converters rsas and tlsrsas as well as the game $\mathbf{G}^{\mathsf{nr\text{-}pca}}$.

Theorem 17 (TLS-RSA)**.** *Let* $\mathcal{C}\subseteq\mathcal{A}_{\mathrm{TCP}}$ *be a set of clients. The TLS-RSA protocol constructs, for each client* $C\in\mathcal{C}$*, one unilaterally secure channel* $\leftarrow\!\!\overset{*}{-}\!\!\twoheadrightarrow\!\bullet_n$ *from* NET*,* PKI*, and* RO_{384}*. Concretely, there exist reductions* $\mathbf{C}$ *and* $\mathbf{C}'$ *such that:*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{tlsrsac}^{C}\mathsf{tlsrsas}_{n}{}^{S}\bot^{E}\left[\mathsf{NET},\mathsf{PKI},\mathsf{RO}_{384}\right], \bot^{E}\left[\bigotimes_{C\in\mathcal{C}}[\![\leftarrow\!\!\overset{*}{-}\!\!\twoheadrightarrow\!\bullet_n]\!]^{(C,S/\rho(C))}, \hat{\mathbf{R}}_n^{\rho(\mathcal{C})}\right]\right)$$
$$\leq \binom{|\mathcal{C}|}{2}\cdot 2^{-224} + \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}'}\left(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F}\right),$$

and there are a simulator σ *and reductions* $\mathbf{C}_q$*, for* $q\in\mathbb{N}$*,* $\mathbf{C}'$*,* $\mathbf{C}''$*,* $\mathbf{C}'''$*, and* $\mathbf{C}^{(iv)}$ *such that for each distinguisher* $\mathbf{D}$*,*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{tlsrsac}^{C}\mathsf{tlsrsas}_{n}{}^{S}\left[\mathsf{NET},\mathsf{PKI},\mathsf{RO}_{384}\right], \sigma^{E}\left[\bigotimes_{C\in\mathcal{C}}[\![\leftarrow\!\!\overset{*}{-}\!\!\twoheadrightarrow\!\bullet_n]\!]^{(C,S/\rho(C))}, \hat{\mathbf{R}}_n^{\rho(\mathcal{C})}\right]\right)$$
$$\leq \left(\binom{n}{2}+\binom{|\mathcal{C}|}{2}\right)\cdot 2^{-224} + n\cdot|\mathcal{C}|\cdot\Gamma_q^{\mathbf{DC}_q}(\mathbf{G}^{\mathsf{nr\text{-}pca}}) + \frac{q}{2^{368}} + \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{CR}}) + |\mathcal{C}|\cdot\Delta^{\mathbf{DC}''}\left(\mathsf{prf}\,\mathbf{U}_{384},\mathbf{F}\right)$$
$$+\,|\mathcal{C}|\left(2\cdot\Delta^{\mathbf{DC}'''}\left(\mathsf{rc4}\ \mathbf{U}_{128},\mathbf{U}^{*}\right) + 2\cdot\Gamma^{\mathbf{DC}^{(iv)}}\left(\mathbf{G}^{\mathsf{suf\text{-}cma}}\right)\right).$$

Proof. The proof of the availability condition follows the same scheme as for DH. A similar argument holds for the security condition, except that we use the bounds for RSA for constructing the resource $\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$ (see Lemma 11). $\square$

7 Conclusion

We described a modular decomposition and proof of the three main unilateral key-exchange modes of TLS. While the purpose of most security mechanisms used in the protocol seemed clear, actually performing the decomposition, i.e. finding suitable boundaries for "cutting" the protocol and specifying the assumed and provided guarantees at each layer proved tedious and sometimes impossible. Several design choices complicated the analysis considerably, they also contradict the idea of a modular protocol design. Beyond the well-known issue of using the encryption and MAC keys in the confirmation message, non-modularity appears in passing lower-level protocol details to higher-level protocols, in using "CPA-secure" key-exchange mechanisms without authentication (see [MTC13] for a different approach), and generally in the late and implicit "authentication" of information via the "finished" messages.

References

[ABP+13] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS and WPA. In *USENIX Security Symposium*, 2013.

[AP12] Nadhem J. AlFardan and Kenneth G. Paterson. Plaintext-recovery attacks against datagram TLS. In *Network and Distributed System Security Symposium (NDSS'12)*, 2012.

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy (SP'13)*, 2013.

[Bar04] Gregory V. Bard. Vulnerability of SSL to chosen-plaintext attack. Cryptology ePrint Archive: Report 2004/111, May 2004.

[BFCZ12] Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zălinescu. Verified cryptographic implementations for TLS. In *ACM Transactions on Information and System Security (TISSEC'12), volume 15(1): 3*, 2012.

[BFK+13a] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptopgrahic security. In *IEEE Symposium on Security and Privacy (SP'2013)*, pages 445–469, 2013.

[BFK+13b] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. Proving the TLS handshake secure (as it is). Technical report, 2013. Available at `http://www.mitls.org/downloads/Proving_the_TLS_Handshake.pdf`.

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel Smart, Bogdan Warinschi, and Steve Williams. Less is more: Relaxed yet composable security notions for key exchange. *International Journal of Information Security*, 12(4):267–297, 2013.

[BPB12] Gilles Barthe, David Pointcheval, and Santiago Zanella Béguelin. Verified security of redundancy-free encryption from Rabin and RSA. In *ACM Conference on Computer and Communications Security*, pages 724–735, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. IACR, Springer, 1993.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001. Extended version in [Can05].

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, December 2005.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of EUROCRYPT'01*, pages 453–474, 2001.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *Proceedings of EUROCRYPT'02*, pages 337–351, 2002.

[CMT13] Sandro Coretti, Ueli Maurer, and Björn Tackmann. Constructing confidential channels from authenticated channels — Public-key encryption revisited. In *Advances in Cryptology — ASIACRYPT 2013*, Lecture Notes in Computer Science, Berlin Heidelberg, 2013. IACR, Springer.

[DR08] Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246, August 2008.

[FHM$^+$12] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In *Proceedings of the ACM Conference on Computer and Communications Security (ACM CCS'12)*, pages 50–61, 2012.

[GIJ$^+$12] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proceedings of the ACM Conference on Computer and Communications Security (ACM CCS'12)*, pages 38–49, 2012.

[GKS13] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. Cryptology ePrint Archive Report 2012/630, 2013.

[GMP$^+$08] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In *Proceedings of ProvSec 2008*, pages 313–327, 2008.

[HFPS99] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet X.509 public key infrastructure. RFC 2459, January 1999. Internet standard, `http://www.ietf.org/rfc/rfc2459.txt`.

[Hic95] Kipp E. B. Hickman. The SSL protocol. Internet draft, February 1995.

[HPFS02] Russell Housley, Tim Polk, Warwick Ford, and David Solo. Internet X.509 public key infrastructure; certificate and certificate revocation list (CRL profile). RFC 3280, April 2002. Internet standard, `http://www.ietf.org/rfc/rfc3280.txt`.

[HSD+05] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *Proceedings of the ACM Conference on Computer and Communications Security (ACM CCS'05)*, pages 2–15, 2005.

[JK02] Jakob Jonsson and Burton S. Kaliski Jr. On the security of RSA encryption in TLS. In *Proceedings of CRYPTO 2002*, pages 127–142, 2002.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In *Proceedings of CRYPTO 2012*, pages 273–293, 2012.

[Kal98] Burt Kaliski. PKCS #7: Cryptographic message syntax. RFC 2315, March 1998. Version 1.5.

[KMO+13] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. Anonymity-preserving public-key encryption: A constructive approach. In E. De Cristofaro and M. Wright, editors, *PETS 2013*, volume 7981 of *LNCS*, pages 19–39, Heidelberg, 2013. Springer.

[KPW13] Hugo Krawczyk, Kenneth Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In *Proceedings of CRYPTO 2013*, pages 429–448, 2013.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communication (or: How secure is SSL?). In *Proceedings of CRYPTO 2001*, pages 310–331, 2001.

[KSS13] Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DH and TLS-RSA in the standard model, 2013.

[KT11] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. Cryptology ePrint Archive, Report 2011/406, 2011.

[Mau02] Ueli Maurer. Indistinguishability of random systems. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132. IACR, Springer-Verlag, 2002.

[Mau11] Ueli Maurer. Constructive cryptography: A new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *TOSCA 2011—Theory of Security and Applications*, Lecture Notes in Computer Science. Springer-Verlag, 2011.

[Mau13] Ueli Maurer. Conditional equivalence of random systems and indistinguishability proofs. In *2013 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 3150–3154, July 2013.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science*. Tsinghua University Press, 2011.

[MSW08] Paul Morrissey, Nigel Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In *Proceedings of ASIACRYPT 2008*, pages 55–73, 2008.

[MT10] Ueli Maurer and Björn Tackmann. On the soundness of Authenticate-then-Encrypt: Formalizing the malleability of symmetric encryption. In *ACM Conference on Computer and Communications Security*. ACM, 2010.

[MTC13] Ueli Maurer, Björn Tackmann, and Sandro Coretti. Key exchange with unilateral authentication: Composable security definition and modular protocol design. Cryptology ePrint Archive, Report 2013/555, 2013. `http://eprint.iacr.org/`.

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap problems: A new class of problems for the security of cryptographic schemes. In *PKC*, volume 1992 of *LNCS*, pages 104–118. Springer, 2001.

[Pau99] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. In *ACM Transactions on Information and System Security (TISSEC)*, pages 332–351, 1999.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology — ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. IACR, Springer-Verlag, 2011. To appear.

[WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.

A More Details on TLS

A.1 X.509 Certificates

In TLS, the server's public key is always certified; optionally, the client may also be required to certify his public key. The certificates used by TLS are X.509 v3 certificates [HPFS02], in practice a chain of certificates starting form the user and ending at a valid certification authority. In practice, each X509 v3 certificate consists of a sequence of three required fields: the `TBSCertificate`, the `AlgorithmIdentifier`, and the `BIT STRING` fields (the latter being a signature). We give more details about the concrete certificate structure below.

In this paper, we abstract the certification process to (user access to) a resource $\mathsf{PKI}_{\mathfrak{F}}$, which takes as input the values that need to be certified, and outputs a certificate, i.e. a bit-string which can be verified. The resource will choose a function $f \in \mathfrak{F}$ and output a certificate of the form $(x, f(x))$, where x is the input that needs to be certified. The function f can be seen as an abstraction of the precise signature generation algorithm. Note that in the X509 certificates the values that are certified are encoded in DER encoding, in particular ensuring that the encoding yields a *unique* value. The encoding consists of a *type*, the *length* of the payload, and the *payload* itself, i.e. the input. We note that this encoding is implicitly assumed in our PKI resource, i.e. the input value x is assumed to correspond to a single certificate. Furthermore, the `tbsCertificate` field contains a unique identifier which bounds each input to a single certificate (per resource), which means, the certificate authority can check whether it has issued the certificate or not.

Finally, we note that in our assessment, we simplify the output of the verification of a certificate to a single bit, i.e. the certificate is valid or invalid. This is a simplification since in fact the output usually yields more information than this: there is a difference between e.g. a valid, but revoked certificate, a certificate which has expired, and an invalid certificate.

We proceed to describe the concrete structure of the X.509 v3 certificates. There are three main fields: `TBSCertificate`, `AlgorithmIdentifier`, and `BIT STRING`. The field `TBSCertificate` consists of:

- **Version.** The version of the certificate, i.e. 1, 2, or 3. If the version is 1, the value is omitted. Already in version 2, the certificates had a unique identifier.
- **Serial number.** This is a unique serial number, which is a positive integer, 160 bits long.
- **Signature.** This field contains the signing algorithm, which must be the same as the one specified in the `AlgorithmIdentifier` field; it may also contain additional parameters.
- **Issuer.** The name of the issuing identity, specified as a sequence of attributes such as country, organization, state or province name, common name, serial number, etc.
- **Validity.** A sequence of two dates, a start and an end date for the validity.
- **Subject.** This field contains the (unique per each issuing CA) identifier of the owner of the public key to be certified.
- **Subject public key info.** This sub-field contains the public key and the algorithm with which the key should be used (e.g. RSA or Diffie-Hellman).
- **Unique identifiers.** These are the subject and issuer unique identifiers, which allow the certification to handle repetitions of either the subject or the issuer fields.
- **Extensions.** The certificates generated in version 3 feature this field, which consists of a sequence of one or more certificate extensions. Examples of such extensions are: subject key identifier (which allows to identify certificates containing a certain public key), key usage (which restricts the use of the key for particular purposes, such as digital signatures, key agreement, etc.), subject alternative name (enabling additional identities to be bound to the subject of the certificate), etc.

The `AlgorithmIdentifier` value contains the algorithm used to generate the signature. The third field, `BIT STRING` is the signature over the DER encoding of the `TBSCertificate` field.

A.2 RSA PKCS#7

A public-key encryption (PKE) scheme with message space $\mathcal{M}$ is typically described as three algorithms $PKE = (gen, enc, dec)$. The key-generation algorithm gen outputs a key pair (pk, sk), the (probabilistic) encryption algorithm enc takes a message $m \in \mathcal{M}$ and a public key pk and outputs a ciphertext $c = enc(m; pk)$, and the decryption algorithm takes a ciphertext c and a secret key sk and outputs a plaintext $m = dec(c; sk)$. It is possible that the output of the decryption algorithm is the special symbol $\perp$; this indicates that the ciphertext c is invalid.

Below, we describe the RSA algorithms following the standard PKCS#7 [Kal98]. Let $\lambda_0 = \Theta(\lambda)$, $\lambda_1 = \Theta(\lambda)$ with $\lambda_0 \leq \lambda_1 - 88$. It is also assumed that λ_1 is a multiple of 8. Strictly speaking, PKCS#7 does not specify a key-generation algorithm gen, but assumes that some RSA key pair is already available. Hence, all our security statements with respect to RSA-based cipher suites are implicitly parametrized by the actual algorithm gen that was used to generate the server's key pair. Consider the following triple of algorithms $RSA = (gen, enc, dec)$.

- $gen(\lambda)$: Upon input the security parameter λ, output $(pk, sk) \coloneqq ((M, e), d)$ such that $ed \equiv 1 \pmod{\phi(M)}$ and the modulus M has λ_1 bits.
- $enc(m; pk)$: Upon input a λ_0-bit message m, pick a random padding $P \in \{0,1\}^{\lambda_1 - \lambda_0 - 24}$ (such that none of the bytes of P equal '00' in hexadecimal notation), define $x \coloneqq 00||02||P||00||m$, and output $c = x^e \pmod{M}$.

- $dec(c; sk)$: Upon input c, attempt to parse $c^d \pmod{M}$ as a sequence of bytes of the form $00||02||P||00||m$ such that P contains no zero bytes and m has exactly λ_0 bits. If the attempt is successful output m; otherwise output a special symbol $\perp$.

When used within TLS, λ_0 is fixed to 384 (yielding a pre-master secret of 48 bytes), while λ_1 is typically 1024 or 2048.

OW-PCA and NR-PCA. The RSA-PKCS-based version of the TLS protocol uses the above defined PKE scheme. Earlier works analyzing the security of TLS-RSA relied on the assumption that RSA-PKCS satisfies a special property called *One-Wayness under Plaintext Checking Attacks* (OW-PCA). This notion is reviewed in Appendix C.1. Not much is known on the validity of this assumption for RSA PKCS#7. The only relevant paper trying to justify it is [JK02], via a reduction to an RSA-like assumption (a.k.a. partial-domain RSA with decision oracle).[18]

Similar to [BFK$^+$13b], our security proof for RSA-TLS relies on a stronger assumption called *Non-Randomizability under Plaintext Checking Attacks.* As in the case of OW-PCA not much is known on the validity of this assumption for RSA PKCS#7. See also Appendix C.1 for a discussion.

A.3 Key Expansion

The key expansion procedure used in TLS relies on a pseudo-random function (based on HMAC). This PRF with the SHA-256 hash function is used for all cipher suites defined in TLS 1.2 (see Section 4). To expand the key, the following function is defined taking as input a secret, a seed, and an identifying label (and produces an output of arbitrary length):

$$\begin{aligned}\mathtt{P}_{\text{hash}}(\text{secret}, \text{seed}) = \mathtt{HMAC}_{\text{hash}}(\text{secret}, \text{A}(1) + \text{seed}) + \mathtt{HMAC}_{\text{hash}}(\text{secret}, \text{A}(2) + \text{seed})\\ + \mathtt{HMAC}_{\text{hash}}(\text{secret}, \text{A}(3) + \text{seed}) + \cdots\end{aligned}$$

where + indicates concatenation. The function $\text{A}(\cdot)$ is defined as: $\text{A}(0) = \text{seed}$ and $\text{A}(\text{i}) = \mathtt{HMAC}_{\text{hash}}(\text{secret}, \text{A}(\text{i} - 1))$. Note that $\mathtt{P}_{\text{hash}}$ can be iterated as many times as necessary to produce the required quantity of data.

B Further Notation and Preliminaries

This section contains further definitions which have been deferred from the main body of the paper. The following Lemma, copied from [Mau13], states that if two games are equivalent, the probability of winning is the same.

Lemma 18 (Mau13, Lemma 1)**.** *If* $\mathbf{S} \overset{g}{\equiv} \mathbf{T}$*, then for any system* $\mathbf{D}$ *and any* q*,*

$$\Gamma_q^{\mathbf{D}}(\mathbf{S}) = \Gamma_q^{\mathbf{D}}(\mathbf{T}).$$

More importantly, the following lemma states that if two systems are equivalent as games, then the distinguishing advantage is upper bounded by the probability of winning the games. This lemma, which originates from [Mau02], is instrumental for many of our proofs.

Lemma 19 (Mau13, Lemma 2)**.** *Let* $\mathcal{A}$ *be a MBO. If* $\mathbf{S}^{\mathcal{A}} \overset{g}{\equiv} \mathbf{T}^{\mathcal{A}}$*, then, for any distinguisher* $\mathbf{D}$ *and any* q*,*

$$\Delta_q^{\mathbf{D}}(\mathbf{S}, \mathbf{T}) \leq \Gamma_q^{\mathbf{D}}(\mathbf{S}^{\mathcal{A}}).$$

[18]It is not clear whether the result of [JK02] applies to RSA PKCS#7 with typical parameters as used in TLS. However we remark that, since its introduction, no weaknesses on the assumption have been reported either.

The following lemma is also needed by some of our proofs. The intuitive interpretation is as follows: For a tuple of games $\mathbf{G}_1, \ldots, \mathbf{G}_n$ which have individually defined MBOs but are equivalent as games *with respect to the disjunction of their MBOs*, the sum of advantages of winning the individual games is at least as large as the advantage for provoking the disjunction.

Lemma 20. *Let $\mathbf{G}_1, \ldots, \mathbf{G}_n$ be a family of random systems (i.e., each $\mathbf{G}_i$ is described by a family of $\mathsf{p}^{\mathbf{G}_i}_{Y^q|X^q}$ for $q \geq 1$) and $\mathcal{A}^1, \ldots, \mathcal{A}^n$ be a family of monotone binary outputs defined on these systems such that A^i_q is independent of Y_q. For $i = 1, \ldots, n$, we write each monotone output as $\mathcal{A}^i = (A^i_1, A^i_2, \ldots)$. Define $\mathcal{A} = \bigvee_{i=1}^n \mathcal{A}^i$, with $\mathcal{A} = (A_1, A_2, \ldots)$ and $A_q = \bigvee_{i=1}^n A^i_q$ (i.e., A_q becomes 1 as soon as there exists a monotone output $\mathcal{A}^i$ whose q^{th} component is 1). Assume that $\mathbf{G}^{\mathcal{A}}_i \overset{g}{\equiv} \mathbf{G}^{\mathcal{A}}_j$ for all $1 \leq i, j \leq n$. Then, for all adversaries $\mathbf{A}$,*

$$\sum_{i=1}^n \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\mathcal{A}_i}_i) \;\geq\; \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\mathcal{A}_i}_{i^*}),$$

for any $1 \leq i^* \leq n$.

Proof. We define the monotone binary outputs $\tilde{\mathcal{A}}^1, \ldots, \tilde{\mathcal{A}}^n$ as

$$\tilde{A}^i_q = \tilde{A}^i_{q-1} \vee \left(A^i_q \wedge \neg \left(\bigvee_{j \neq i} A^j_{q-1} \right) \right),$$

i.e., $\tilde{\mathcal{A}}^i$ formalizes that $\mathcal{A}^i$ is (among) the *first* MBOs to become 1. Indeed, $\tilde{A}^i_q = 1$ if either: (1) the MBO $\tilde{\mathcal{A}}^i$ had already turned 1, i.e., $\tilde{A}^i_{q-1} = 1$; (2) the q^{th} value A^i_q is 1, but no previous value A^j_{q-1} of any other MBO $\mathcal{A}^j$ is 1. Still, $\mathcal{A}$ becomes 1 as soon as any one of the outputs A^i_q becomes 1 for some i, q. If this output becomes 1, there must be at least one $\tilde{A}^i_q$ that became 1 first. Thus, $\mathcal{A}$ becomes 1 if and only if at least one $\tilde{A}^i_q$ becomes 1, yielding $\mathcal{A} = \bigvee_{i=1}^n \tilde{\mathcal{A}}^i$. Let now $1 \leq i^* \leq n$. It holds that: $\sum_{i=1}^n \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\tilde{\mathcal{A}}_i}_{i^*}) \geq \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\mathcal{A}}_{i^*})$, since $\mathcal{A}$ is triggered if and only if it was triggered first in one of the $\tilde{A}_i$ outputs. On the other hand, we also have $\Gamma^{\tilde{\mathcal{A}}_i}_q(\mathbf{G}_i) = \Gamma^{\tilde{\mathcal{A}}_i}_q(\mathbf{G}_j)$ since provoking $\tilde{\mathcal{A}}$ implies provoking $\mathcal{A}$ while $\mathbf{G}_i$ and $\mathbf{G}_j$ are still behaving equivalently (since, if $\tilde{\mathcal{A}}$ was not triggered before, this means that $\mathcal{A}$ does not hold). Hence, we obtain

$$\sum_{i=1}^n \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\mathcal{A}_i}_i) \;\geq\; \sum_{i=1}^n \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\tilde{\mathcal{A}}_i}_i) \;=\; \sum_{i=1}^n \Gamma^{\mathbf{A}}_q(\mathbf{G}^{\tilde{\mathcal{A}}_i}_{i^*}) \;\geq\; \Gamma^{\mathbf{A}}(\mathbf{G}^{\mathcal{A}}_{i^*}),$$

which concludes the proof. □

B.1 Signature Schemes

A signature scheme is a triple of algorithms $\mathit{SIG} = (\mathit{gen}, \mathit{sign}, \mathit{vrf})$. The *key-generation* algorithm *gen* takes no input[19] and outputs a pair $(\mathit{sk}, \mathit{vk})$ of a *signature key sk* and a *verification key vk*. The *signing* algorithm *sign* takes as input a signature key *sk* and a message $m \in \mathcal{M}$ of some message space $\mathcal{M}$, and outputs a signature $s = \mathit{sign}(\mathit{sk}, m)$. The (often deterministic) *verification* algorithm *vrf* takes as input a verification key *vk*, a message m, and a signature s, and outputs a decision bit. A signature scheme is correct if for any key pair $(\mathit{sk}, \mathit{vk})$ generated by *gen* and for all $m \in \mathcal{M}$, $\mathit{vrf}(\mathit{vk}, m, \mathit{sign}(\mathit{sk}, m)) = 1$.

The common security requirement for a signature scheme $\mathit{SIG} = (\mathit{gen}, \mathit{sign}, \mathit{vrf})$ is called *unforgeability* and is formalized in Section C.4.

using the following game $\mathbf{G}^{\mathit{SIG}}$:

[19]For an asymptotic treatment, the algorithm takes as input the security parameter.

1. Generate a key pair $(sk, vk) = gen()$ and output vk to the adversary.
2. (Repeatedly) Given a message $m \in \mathcal{M}$ from the adversary, compute $s = sign(sk, m)$, store m in an internal buffer $\mathcal{B}$, and return s to the adversary.
3. Upon input a pair (m', s') with $m' \notin \mathcal{B}$ and $vrf(vk, m', s') = 1$, output that the game is won.

For $\varepsilon \in [0, 1]$, a signature scheme is ε-secure with respect to a class $\mathcal{D}$ of adversaries if $\Gamma^{\mathbf{A}}(\mathbf{G}^{SIG}) \leq \varepsilon$ for all $\mathbf{A} \in \mathcal{D}$.

C Game-based Definitions

This Appendix collects the relevant game-based definitions that are used in our analysis of TLS.

Game-based definitions specify a property of a cryptographic scheme based on an interaction between two (hypothetical) entities: the game (or challenger) and the adversary. During the interaction, the adversary may issue "oracle queries" to the challenger, the responses of which model what information may be leaked to the adversary. The adversary's goal is specified by the game, and could be, e.g., forging a message or distinguishing encryptions of different messages. If this game cannot be won by any (efficient) adversary, then the scheme is secure against the considered type of attack. The formal definition of a game is given in Definition 5.

Bit guessing games Some games in the literature are *bit-guessing games*. These games can often be described by a pair of systems $\mathbf{G}_0$ and $\mathbf{G}_1$, with the interpretation that in the beginning of the game, a bit $B \in \{0, 1\}$ is chosen uniformly at random. The adversary will then be given access to $\mathbf{G}_B$, and the goal is to guess the bit B. The adversary can win such a game with probability $\frac{1}{2}$ trivially by simply guessing the hidden bit. Hence, we measure the adversary's success in terms of his *advantage*, that is, the (absolute) difference between $\mathbf{A}$'s probability of winning $\mathbf{G}$ and the success probability for these "trivial" strategies, formally $\Phi^{\mathbf{A}}(\mathbf{G}) = 2 \cdot \left|\Gamma^{\mathbf{A}}(\mathbf{G}) - \frac{1}{2}\right|$. Note also that $\Phi^{\mathbf{A}}(\mathbf{G}) = \Delta^{\mathbf{A}}(\mathbf{G}_0, \mathbf{G}_1)$.

C.1 OW-PCA & NR-PCA

We review the notion of one-wayness against plaintext checking attacks [JK02]. Let $PKE = (gen, enc, dec)$ be a public key encryption scheme with message space $\mathcal{M}$.

Consider the game of Figure 6.

```
Init()
$(pk, sk) \leftarrow gen()$
$\mathsf{Chal} \leftarrow \emptyset$
$W \leftarrow 0$
return $pk$
end.

ChGen()
if $\mathsf{Chal} \neq \emptyset$
    return $\perp$
else
    $m^* \leftarrow \mathcal{M}$
    $\mathsf{Chal} \leftarrow enc(m^*; pk)$
    return $\mathsf{Chal}$
end.

PCA$(m, c)$
if $(m = \perp) \vee (\mathsf{Chal} = \emptyset)$
    return $\perp$
else if $m = dec(c; sk)$
    return 1
else
    return 0
end.

GameOutput$(m')$
if $\mathsf{Chal} = \emptyset$
    return $\perp$
else
    return $\mathsf{Output} = (m' = m^*)$
end.
```

Figure 6: The OW-PCA security game, $\mathbf{G}^{\mathsf{ow\text{-}pca}}$

Definition 21. We say that $PKE = (gen, enc, dec)$ is ε-OW-PCA with respect to a class $\mathcal{D}$ of adversaries if for every $\mathbf{A} \in \mathcal{D}$ it holds that $\Gamma^{\mathbf{A}}(\mathbf{G}^{\mathsf{ow\text{-}pca}}) \leq \varepsilon$.

Definition 21 intuitively says that it is hard to "invert" a ciphertext, even given access to a plaintext checking oracle (i.e., an oracle allowing to check if a guess for the plaintext corresponding to some ciphertext is correct). As discussed in Section 3.3.3, our proof for TLS-RSA also relies on the assumption that it is hard to re-randomize a ciphertext, even given access to a plaintext checking oracle. The latter notion, also known as non-randomizability against plaintext checking attacks (NR-PCA), was recently introduced in [BFK+13b]; the corresponding game is depicted in Figure 7.

```
**Init**()
$(pk, sk) \leftarrow gen()$
$\mathsf{Chal} \leftarrow \emptyset$
$W \leftarrow 0$
return $pk$
**end.**
```

```
**ChGen**()
if $\mathsf{Chal} \neq \emptyset$
    return $\perp$
else
    $m^* \leftarrow \mathcal{M}$
    $\mathsf{Chal} \leftarrow enc(m^*; pk)$
    return $\mathsf{Chal}$
**end.**
```

```
**PCA**$(m, c)$
if $(m = \perp) \vee (\mathsf{Chal} = \emptyset)$
    return $\perp$
else if $m = dec(c; sk)$
    return 1
else
    return 0
**end.**
```

```
**GameOutput**$(c')$
if $\mathsf{Chal} = \emptyset$
    return $\perp$
else
    return $\mathsf{Output} = (c' \neq c \wedge dec(c'; sk) = m^*)$
**end.**
```

Figure 7: The NR-PCA security game, $\mathbf{G}^{\mathsf{nr\text{-}pca}}$

Definition 22. We say that $PKE = (gen, enc, dec)$ is ε-NR-PCA with respect to a class $\mathcal{D}$ of adversaries if for every $\mathbf{A} \in \mathcal{D}$ it holds that $\Gamma^{\mathbf{A}}(\mathbf{G}^{\mathsf{nr\text{-}pca}}) \leq \varepsilon$.

Note that NR-PCA implies OW-PCA whenever the encryption algorithm is randomized, because if we can invert the challenge ciphertext we can also re-randomize it. The other direction might not be true, as there might be easier ways to re-randomize a ciphertext than by inverting it. [BFK+13b] conjectured that the NR-PCA assumption follows from the common-input extractability assumption of [BPB12] and OW-PCA.

In TLS the message space is of the form $\mathcal{M} = \texttt{version\_number} \times \{0,1\}^{368}$. Note that decryption may result in a message that is outside of the message space and that such invalid plaintexts can still be checked using the **PCA** oracle.

C.2 Gap Diffie-Hellman

For some security statements, we use the gap Diffie-Hellman assumption, originally proposed by Okamoto and Pointcheval [OP01], which essentially states that it is hard to compute g^{xy} given (g, g^x, g^y), even given access to a DDH verification oracle $(\cdot,\cdot,\cdot)$. We formalize this problem parametrized by a distribution $\mathcal{G}$ over groups of finite order $|\mathbb{G}| = q$, together with a (publicly-known) generator g. The game is specified in Figure 8.

```
**Init**$(\mathcal{G})$
draw $(\mathbb{G}, g) \leftarrow_{\$} \mathcal{G}$
draw $x, y \leftarrow_{\$} \{1, \ldots, |\mathbb{G}|\}$
return $(\mathbb{G}, g, g^x, g^y)$
set $W \leftarrow 0$
**end.**
```

```
**DDH**$(g^a, g^b, C)$
return $(g^{ab} = C)$
**end.**
```

```
**GameOutput**$(Z)$
$W \leftarrow (Z = g^{xy})$
**end.**
```

Figure 8: The GapDH security game, $\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}$

Definition 23. A group distribution $\mathcal{G}$ satisfies the Gap Diffie-Hellman assumption with respect to the class $\mathcal{D}$ of adversaries and with error ε if for all $\mathbf{A} \in \mathcal{D} : \Gamma^{\mathbf{A}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) \leq \varepsilon$.

C.3 Collision Resistance

For the key expansion step in Section 4, the client and server use a hash function H. In order to prevent the adversary from changing some of the protocol messages without modifying the finished messages ξ_C, ξ_S, this hash function needs to be collision resistant.

```
**GameOutput**$(x, x')$
$W \leftarrow (x = x')$
**end.**
```

Figure 9: The collision resistance game, $\mathbf{G}^{\mathsf{CR}}$

Definition 24. A hash function H is collision-resistant for a class $\mathcal{D}$ of adversaries, if for all $\mathbf{A} \in \mathcal{D}$ it holds: $\Gamma^{\mathbf{A}}(\mathbf{G}^{\mathsf{CR}}) \leq \varepsilon$.

C.4 Unforgeability under Chosen-Message Attacks

The security condition for a signature scheme as defined in Section B.1 is a tuple of algorithms $\mathit{SIG} = (\mathit{gen}, \mathit{sign}, \mathit{vrf})$. The standard security requirement for a signature scheme is *unforgeability under chosen-message attacks (UF-CMA)* and formalizes that no attacker may be able to forge a signature, even if he is given access to a "signature oracle" that returns signatures for arbitrary messages. (Of course, signatures returned by the oracle are not eligible for winning the game.)

Init$()$	**Sign**(m)	**Forge**(m, s)
$(\mathit{sk}, \mathit{vk}) \leftarrow \mathit{gen}()$	$s \leftarrow \mathit{sign}(m; \mathit{sk})$	if $(m \notin \mathcal{B}) \wedge \mathit{vrf}(m, s; \mathit{vk})$
$\mathcal{B} \leftarrow \emptyset$	$\mathcal{B} \leftarrow \mathcal{B} \cup \{m\}$	$W \leftarrow 1$
$W \leftarrow 0$	return s	**end.**
return vk	**end.**	
end.		

Figure 10: The unforgeability game for the scheme SIG, $\mathbf{G}^{\mathsf{uf\text{-}cma}}$.

Definition 25. A signature scheme is existentially unforgeable under chosen message attacks for a class $\mathcal{D}$ of adversaries, if for all $\mathbf{A} \in \mathcal{D}$ it holds: $\Gamma^{\mathbf{A}}(\mathbf{G}^{\mathsf{uf\text{-}cma}}) \leq \varepsilon$.

Resource	Description	Constructed from	Sub-protocol	Security Loss
Basic Resources:				
$- \twoheadrightarrow$	one-way, insecure channel; can read, change, and inject messages at E-interface.	assumed resource		0
$\mathsf{PKI}_{\mathfrak{F}}$	Public-Key Infrastructure; provides certification of PKs.	assumed resource		0
RO_{384}	Random Oracle; outputs consistent randomness.	assumed resource		0
NET	the insecure point-to-point network; parallel composition of $- \twoheadrightarrow$ and $\twoheadleftarrow -$ channels; behave as for $- \twoheadrightarrow$ at E-interface.	assumed resource, $[\![- \twoheadrightarrow, \twoheadleftarrow -]\!]^{(C,S/C)}$ for $C \in \mathcal{A}_{\mathrm{TCP}}$		0
NAME_{ρ}	unique name resource; associates each client interface $C \in \mathcal{C}$ with a unique nonce η, as ρ indicates.		rnd	$\binom{\lvert\mathcal{C}\rvert}{2} \cdot 2^{-224}$
Intermediate TLS Resources				
$\mathsf{SNET}_{\mathsf{N},\rho,n}$	insecure network with nonce exchange; associates each client interface $C \in \mathcal{C}$ with a unique nonce η, as ρ indicates.	$\mathsf{NAME}_{\rho}, \mathsf{NET}$	$\mathsf{hec}, \mathsf{hes}$	$\binom{n}{2} \cdot 2^{-224}$
$\rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$	authenticated transmission network; network with one-sided authentication of the group parameters used in TLS-DHE.	$\mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}$	$\mathsf{vrf}, \mathsf{sgn}$	$\Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{uf\text{-}cma}})$
$\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$	master secret resource; allows parties to obtain the master secret. For each session, MS can either be injected or honestly generated.	TLS-DH: $\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}$	$\mathsf{dhc}, \mathsf{dhs}_{\mathcal{G}}$.	$\Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}})$
		TLS-DHE: $\mathsf{RO}_{384}, \rhd\!\!-\!\bullet_{\mathsf{N},\rho,\mathfrak{F},SIG,n}$	$\mathsf{dhec}, \mathsf{dhes}_{\mathcal{G}}$	$n\lvert\mathcal{C}\rvert \cdot \Gamma^{\mathbf{DC}}(\mathbf{G}_{\mathcal{G}}^{\mathsf{GapDH}}) + \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{uf\text{-}cma}})$
		TLS-RSA: $\mathsf{RO}_{384}, \mathsf{SNET}_{\mathsf{N},\rho,n}, \mathsf{PKI}_{\mathfrak{F}}$	$\mathsf{rsac}, \mathsf{rsas}$	$n\lvert\mathcal{C}\rvert \cdot \Gamma_q^{\mathbf{DC}_q}(\mathbf{G}^{\mathsf{nr\text{-}pca}}) + \frac{q}{2^{368}}$
Final TLS Resources:				
$\overset{KSP,*}{=\!=\!\bullet}$	unilaterally authenticated key; can inject keys or allow them to be honestly distributed.	$\mathsf{MSK}_{\mathsf{N},\rho,AUX,n}$	$\mathsf{expc}, \mathsf{exps}$	$\lvert\mathcal{C}\rvert \cdot \Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathsf{PRF}}) + \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{CR}})$
$\twoheadleftarrow \overset{*}{\twoheadrightarrow}\bullet$	Unilaterally secure 2-party communication. For each session, adversary can either interfere or not.	$\overset{KSP,*}{=\!=\!\bullet}$	stream cipher	$2 \cdot \Delta^{\mathbf{DC}}(\mathsf{rc4}\ \mathbf{U}_{128}, \mathbf{U}^*) + 2 \cdot \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{suf\text{-}cma}})$
			CBC	$2 \cdot \Delta^{\mathbf{DC}}(\mathsf{bc}\ \mathbf{U}_k, \mathbf{P}_\ell) + 2 \cdot \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathsf{suf\text{-}cma}}) + \frac{(ql)^2}{2^{\ell-1}}$

Figure 11: The resources used in this work